Tool Integration: Model-based Tool Adapter Construction and Discovery Conforming to OSLC

Wenqing Gu
Abstract

**Tool Integration: Model-based Tool Adapter Construction and Discovery Conforming to OSLC**

Wenqing Gu

Tool Integration is a vital part in the modern IT industry. With the ever increasing complexity in reality, more tools in different domains are needed in research and development process. However, currently no vendor has a complete solution for the whole process, and no mature solution to integrate different tools together, thus tools are still used separately in the industry. Due to this separation, the same information is created more than once for different tools, which is both time wasting and error prone.

This thesis is part of the research to deliver a model-based tool integration framework that helps the end user design their own scenario of tool integration and implement it with less effort by generating most common parts automatically. This thesis itself is mainly focused on tool adapters, including the model-based tool adapter construction and discovery. In the first part, a model-based tool adapter construction platform conforming to OSLC is designed and implemented, based on which, the construction process of a tool adapter is presented with an example. With this platform, most of the codes and configuration files can be generated, with the exemption of the tool specific functionalities. The tool adapter is constructed as a separate SCA component, and can be included in the SCA based tool chain with minor configuration. With SCA, the deployment of the tool adapter and future management can be largely eased. In the second part, the model-based discovery process of an unknown tool adapter conforming to OSLC and our assumptions is presented in detail. With the discovery tool, the sharing of the tool adapter is made possible, and the integration of the different tools are largely eased. An example of discover an unknown tool adapter is also included for a more clear explanation.

Finally, in the meanwhile of the design and implementation of the construction platform and the discovery process, the existing Matlab/Simulink tool adapter is extended and refined to make it full compatible to the standard and our tool chain.
Acknowledgements

First of all, I would like to show my greatest gratitude to my supervisor Matthias Biehl and reviewer Frederic Loiret for always giving me nice suggestions and great ideas during the six months thesis time in KTH. Without them, I can’t finish this thesis alone.

Secondly, I would like to thank Dr. Qin Liu in Tongji University for her kind help in the thesis and great support in participating this Sino-Swedish Master Programme in Computer Science and Software Engineering.

Finally, I would like to thank my girl friend, and all my friends that helped in the thesis.
Contents

1 Introduction .................................................................................................................. 1
  1.1 The Problem ......................................................................................................... 1
  1.2 The Context of this Thesis Work ....................................................................... 2
  1.3 Major Work and Contributions ......................................................................... 2
  1.4 Thesis Structure ................................................................................................ 3

2 State of the Art ........................................................................................................... 4
  2.1 Introduction .......................................................................................................... 4
  2.2 Modern Practices on Tool Integration .............................................................. 4
    2.2.1 OSLC and The Jazz Platform ...................................................................... 4
    2.2.2 MOFLON and Related Tool Integration Efforts ....................................... 5
    2.2.3 Other Integration Solutions ...................................................................... 6
  2.3 Related Work on Tool Discovery ....................................................................... 7
    2.3.1 Current Status on Tool Discovery ................................................................ 7
    2.3.2 Service Discovery and Tool Discovery ...................................................... 8

3 Technical Background ............................................................................................. 10
  3.1 Introduction .......................................................................................................... 10
  3.2 Eclipse Modeling Framework, EMF .................................................................. 10
    3.2.1 EMF and the Ecore Model ...................................................................... 10
    3.2.2 Acceleo ................................................................................................... 11
  3.3 Open Services for Lifecycle Collaboration, OSLC ............................................ 12
    3.3.1 OSLC and REST .................................................................................. 12
    3.3.2 OSLC and RDF ................................................................................... 13
    3.3.3 Implementation Details: JAX-RS, Jersey, Apache CXF and FreeMarker ................................................................................................. 14
  3.4 Service Component Architecture, SCA ............................................................... 14
    3.4.1 SOA and SCA ......................................................................................... 14
6.2.1 Model Design of the Discovered Tool Adapter ......................... 47
6.2.2 The Discovery Algorithm .............................................. 48
6.2.3 Saving the Discovered Model ........................................ 58
6.2.4 Model-based Generation of Code Stubs ............................. 58

6.3 Summary of Assumptions .................................................. 61

7 Conclusion ............................................................................. 63
7.1 Summary ............................................................................ 63
7.2 Limitations ......................................................................... 63
7.3 Future Work ........................................................................ 64

References .................................................................................. 65

Appendix A: Generated Code Stubs for Matlab/Simulink Adapter Based on the Tool Adapter Construction Platform .................................................. 68
List of Tables

Table 2.1: Compatibility between Discovery and Orchestration Methods .................. 9
Table 5.1: Parameters predefined in the construction platform .............................. 28
Table 5.2: Different output styles of XMI-based and RDF-based output services .... 30
Table 5.3: Data Services to be provided in Matlab/Simulink .............................. 32
Table 5.4: Control Services to be provided in Matlab/Simulink ............................. 32
Table 5.5: Limitations and assumptions of the tool adapter construction platform . 44
Table 5.6: Limitations and assumptions of the current Matlab/Simulink implementation ................................................................................................................. 45
Table 6.1: Assumptions made in the automated discovery algorithm of unknown tool adapter conforming to OSLC ................................................................. 62
List of Figures

Figure 1.1: General idea of the model-based tool integration framework .......................... 2
Figure 3.1: Simplified structure of Ecore meta model [32] ........................................... 11
Figure 3.2: OSLC core specification concepts and relationships [34] ......................... 13
Figure 3.3: SCA example with multiple composites and components running on multiple machines [7] ................................................................. 15
Figure 3.4: SCA component structure [7] .................................................................. 16
Figure 3.5: SCA component interaction example [7] .................................................... 16
Figure 3.6: A simple Helloworld SCA application running in FraSCAti explorer ....... 18
Figure 3.7: Architecture of the simple Helloworld SCA application ......................... 18
Figure 5.1: Control flow for OSLC-conform tool adapter design based on the construction platform ................................................................. 24
Figure 5.2: Design of the ifest-common model ........................................................... 25
Figure 5.3: Composite Diagram of the Matlab/Simulink Tool Adapter ....................... 26
Figure 5.4: Design of the data model of Matlab/Simulink Tool Adapter .................... 34
Figure 5.5: Design of the core model of Matlab/Simulink Tool Adapter .................... 34
Figure 5.6: Java implementation generated of the data and core model of Matlab/Simulink tool adapter ................................................................. 35
Figure 5.7: Tool adapter stubs generated based on the data and core model of Matlab/Simulink tool adapter ................................................................. 35
Figure 5.8: Architecture of Matlab/Simulink Tool Adapter (Manual Part) ............... 36
Figure 5.9: Screen shot of Matlab/Simulink model york ............................................ 38
Figure 6.1: General process of automated discovery of unknown tool adapters conforming to OSLC ................................................................. 47
Figure 6.2: Detail algorithm of the automated discovery process ................................. 48
Figure 6.3: Discovered data model of Matlab/Simulink Tool Adapter ..................... 59
Figure 6.4: Discovered core model of Matlab/Simulink Tool Adapter ..................... 60
1. Introduction

1.1 The Problem

With the ever increasing complexity of the modern products, a growing number of factors should be considered during the development process. In the embedded industry, experts from different domains usually work together to design different aspects of the products. These activities, including the hardware design, the software design, the requirement engineering, coding and verification, all rely heavily on software development tools.

However, the tools supporting different parts of the work are usually not integrated, despite the needs to be used as a tool chain to exchange the information and provide traceability in different parts. Whenever information exchange between two tools is needed, the engineers have to manually create duplicate information in another different format for the other tool. This redundant work is not only a wasting of time, but also one of the important factors that lead to the defects in the final products. Also, without the support of the treatabilities of the work products in different parts of the whole process, the validation will be difficult.

The tool integration is vital to the industry. However, the tools in the development process are targeted to totally different areas, therefore it will be difficult for one company to provide a complete solution for the whole process. Also, many of the tools are quite famous in its own area, and have been used for quite a long time, thus, it is unlikely that the vendors agree on a new standard and change to it in a very short time. Therefore, to conclude, the only reasonable solution to create a usable integrated solution is to integrate the tools externally.

In recent years, various efforts have been made to study the integration of different tools. One of the styles is to expose the internal data and other functionalities as web services, and orchestrate them as a whole tool chain. The integration is done in an external way with tool adapters created as a bridge between the services and the tools, where tool adapters are only used to provide a unite service interface and real operations are done by calling APIs of the tools as a black box.

With this simple idea, our research team is trying to propose a RESTful [11] service oriented integration solution following the becoming industry standard for tool life-cycle integration OSLC (Open Services for Lifecycle Collaboration)[34]. In the aspect of tool adapters, studies to further ease the work to develop a new tool adapter and create an ecosystem of sharing of tool adapters from different vendors should be conducted.
1.2 The Context of this Thesis Work

In the big picture, the idea is to create a model-based tool integration framework. More specifically, TIL model is introduced [5] as the glue of the different kinds of tool adapters with different service models in the tool chain and a model-based generation engine is developed to generate the code stubs for the whole tool chain. In the meanwhile, the guidelines to construct the new tool adapters are designed, which the developer of the tool adapter must follow. With the same guideline, the development of tool adapters can be largely eased, and the sharing of tool adapters from different vendors is made possible. An unknown tool adapter discovery tool is also designed and implemented to facilitate the integration of the tool adapters from others. This idea in the big picture is presented in Figure 1.1.

![Figure 1.1. General idea of the model-based tool integration framework](image)

1.3 Major Work and Contributions

In the research, this thesis is mainly focused on tool adapters, including the tool adapter construction and discovery conforming to OSLC. More specifically, the work includes:

- **Extend an existing tool adapter generator**: One aspect of the work is to develop a tool adapter construction platform based on the existing tool adapter generator. Firstly the guidelines of OSLC-conform tool adapter model are created, and based on the model following the guidelines, model-to-tool-adapter generation engine is also developed to generate the stubs of the tool adapter. Besides, supporting components like the *ID Resolving Service* to provide uniform IDs for different tools and the *Converter Service* to output the responses in the standard way according to OSLC are also developed.

- **Newly developed tool adapter discovery tool**: In the second part of the work, automated discovery of an unknown tool adapter conforming to OSLC and our assumptions is studied. As in OSLC core specification [34], with the URI of *ServiceProviderCatalog* as the start point of the discovery process, data and...
control services of the tool adapter can be discovered and stored in the discovered tool adapter model. Based on this model, similar model-to-tool-adapter generation engine is developed to generate the code stubs needed to consume the services provided.

- **Improve the existing Matlab/Simulink tool adapter:** In the meanwhile of developing the tool adapter construction platform and discovery tool, the existing Matlab/Simulink tool adapter is extended and refined to be fully compatible to the standard and our tool chain.

The contributions of the thesis are mainly in the standardization of tool adapter modeling, the development of the tool adapter construction platform, as well as the study of the automated discovery process of an unknown tool adapter conforming to OSLC and our assumptions.

### 1.4 Thesis Structure

The rest of this thesis report is structured as follows:

**In Chapter 2**, we present the state of the art in tool integration and tool discovery. In the first part, modern integration solutions are discussed. In the second part, research efforts concerning tool adapter discovery as well as service discovery in general are analyzed and compared.

**In Chapter 3**, we introduce all the technologies that have been used in the thesis work in general. In this section, the technologies related with each other are introduced together as the same group.

**In Chapter 4**, we first present the user scenarios of the tool adapter construction and discovery. And then, the iterative development process of the thesis is introduced in general.

**In Chapter 5**, we present the construction process of the tool adapter. First, in the aspect of building the tool adapter construction platform, we introduce the design guidelines of the tool adapter model, the architecture of the tool adapter to be generated and the model-to-tool-adapter generation engine. Then, in the aspect of tool adapter construction with this platform, with the example of creating a Matlab/Simulink tool adapter, we present the whole process including the model design, the generation of code stubs, the implementation of the tool adapter and the final testing.

**In Chapter 6**, we present the discovery process of the unknown tool adapter conforming to OSLC and our assumptions, including the model design of the discovered tool adapter, the discovery algorithm, the generation of the discovered model and the model-to-tool-adapter generation. The introduction is done step by step with an example to discover the unknown Matlab/Simulink tool adapter.

**In Chapter 7**, we conclude with a summary of the thesis work as well as the discussion of the major limitations and possible future work.
2. State of the Art

2.1 Introduction

Tool integration is defined as the process to produce an effective and integrated automated environment that supports the complete software development life cycle [6][35][36].

As summarized in [39][38][6], since 1980s, there’re various research efforts concerning this topic including concepts, models, approaches, techniques and mechanisms in support of integration, etc. Also, to facilitate the tool integration, a number of technologies have been used including old technologies like CORBA [24] and new technologies like web services in SOAP (Simple Object Access Protocol) [13] or REST.

In early approaches, the integration solutions are mostly ad-hoc which require a lot of redundant work in dealing with different integration problems. Therefore, to save the manual efforts, in the modern solutions, tool integration platforms are created and applied with which the reuse of the common integration functionalities is made possible.

In the following part, we first introduce some of the modern practices on tool integration, including OSLC and the Jazz platform, MOFLON and related solutions, and some other frameworks or platforms. After that, efforts on tool adapter discovery as well as service discovery in general are introduced.

2.2 Modern Practices on Tool Integration

2.2.1 OSLC and The Jazz Platform

OSLC is one of the major efforts in the domain of tool integration, and it is the becoming industry standard in this area. It is not a technical framework or platform, but a standard on the way that software lifecycle tools can share data with one another, as introduced on its website1.

In one part, OSLC adopts RESTful web service standard, and defines services that are shared with other tools as resources and publishes them as RESTful web services. Also because of the lack of available directory services in the RESTful web service standard, OSLC defines directory services to make the discovery of the services provided possible.

1http://open-services.net/
In the other part, in OSLC, industry experts from different domains are working together to design and agree on the scenarios and resources needed to support the common tasks and interactions in software tools. OSLC has created a set of open and public descriptions of resources and service interfaces in different common scenarios that can be and have already been adopted by tool vendors including change management, quality management, requirements management, software project management, product lifecycle management, etc.

Jazz [12] is one of the modern tool integration platforms with the whole architecture built on OSLC. Similar to the team’s research work, different tools share data related functionalities through OSLC resources, and the Jazz platform is the core part to coordinate the service sharing.

Different to the team’s research work, the Jazz platform begins with the needs of team collaboration and project management in software development, and it is itself a team collaboration tool. By attaching different tools that share functionalities in OSLC, more information can be accessed and tracked in the tool. In the context of this thesis project, tool integration is the key problem which is more general compared to Jazz, and this thesis project is mainly concentrating on the development and discovery of the tool adapters that help the work of tool integration.

Also, unlike the heavy concentration we have paid in the team’s research work and this thesis project on how to help the construction of tool adapter in an easy way, Jazz is more like a tool that support IBM’s Rational Unified Process\(^2\) and to unify other Rational products made by IBM.

In the integration part, Jazz is more concentrating on the data sharing between the tools, as in one part the OSLC currently only supports data resources smoothly and in the other part only data from different tools are interesting to a team collaboration tool. In the scenario of development with a set of tools, certainly the integration of control functionalities are needed.

Jazz is an excellent team collaboration tool with full integration support of IBM’s Rational products. However, as an open community\(^3\), Jazz is still underway. Because of the lack of support in developing own products that could be integrated to Jazz, right now only a few IBM’s products can be integrated to Jazz. In the future, with the development of the community, similar to Eclipse, probably Jazz will become the industrial standard platform for team collaborations with the capability to integrate tools developed by different vendors.

2.2.2 MOFLON and Related Tool Integration Efforts

MOFLON\(^4\) is another framework for metamodeling and model transformation similar to EMF (Eclipse Modeling Framework)[32] we adopted in this thesis. It is a pioneer in the modeling world, as introduced in [37]. The development of MOFLON began

\(^2\)http://www-01.ibm.com/software/awdtools/rup/
\(^3\)http://jazz.net
\(^4\)http://www.moflon.org/
in 2002, and four years later the first version was released. In 2007 and 2008, more components including a model-to-model transformation editor and generator, a compiler for the Object Constraint Language (OCL) [21], and modularization concepts for model-to-model transformations are included in later versions.

The main idea of MOFLON is to adopt MOF 2.0 standard [21] as the metameta-model to represent the tool metamodel. Existing modeling environment for UML models is refined with a plugin provided by MOFLON to provide the metamodel modeling environment. The code generated from metamodels complies to the JMI standard [33] through MOFLON’s own extension on the basis of an existing JMI mapping for MOF 1.4 defined by the Object Management Group (OMG).

Based on MOFLON, there’re a few tool integration efforts, however due to the limitations of MOFLON, those efforts are only on the basis of data integration, or more specifically on the basis of the transformation of the data from different tools.

As presented in [2] and [37], with the help of MOFLON and tool adapters, requirements engineering tool DOORS and the system modeling environment MATLAB/Simulink can be integrated. Similar to this thesis work, the tool adapters can partly be generated by MOFLON on the basis of metamodeling of the tool to provide a standardize JMI compliant interface to be used in the model analysis and transformation.

However, because of the use of JMI, only data resources can be extracted and operated on the JMI interfaces, which will only work on data integration of the tools. Besides, without the support of directory services of the resources provided, intelligent algorithm for the resource discovery and transformation can be difficult to design, and more manual work required to specify those constraints and rules to analyze or transform the model. Finally, the most important, the standards MOFLON adopted are becoming obsolete, that’s also the reason that MOFLON is under re-engineering as reported in [3] to support new de facto standard like EMF (Eclipse Modeling Framework).

2.2.3 Other Integration Solutions

There are a few other integration solutions like ModelBus5, Atego Workbench6 or Eclipse7. However, they all have some kinds of drawbacks or major limitations to be used to integrate existing tools.

ModelBus is a tool integration technology which is built on Web Services and follows a SOA approach [16]. The architecture is deployed on SOAP web services. Clients are divided into service providers and consumers in a model bus, both of which should create an adapter to integrate into Apache CXF DOSGi8 used in ModelBus.

---

5http://www.modelbus.org/modelbus/
6http://www.atego.com/products/atego-workbench/
7http://www.eclipse.org/
8http://cxf.apache.org/dosgi-releases.html
ModelBus pays heavy attention to the models created by tools. There’s also a central repository included as an infrastructure service in ModelBus to be used as the media to share the models between tools to provide the traceability of different models between tools.

Despite the functionalities provided by ModelBus, we can find the following drawbacks:

- ModelBus pays too much attention on its central repository and the traceability functionality of the models from different tools. For other integration tasks like the control integration, it lacks basic support.
- ModelBus does not provide any support to ease the development of tool adapter. Tool adapter should be created manually, which is quite time consuming.
- ModelBus does not have a reliable SOA platform as the running container of the servers, instead, the services are deployed directly on application servers like Tomcat\(^9\). In this case, the services of the tool adapters will be difficult to manage.

Atego Workbench, on the other hand, is based on thin client architecture and the server technology. In Atego, the servers are the ones that are really running the applications, whereas, the clients are only presenting the UI and results from the server. Because of the adoption of the special architecture, the user can reduce cost of either the deployment or the software licenses. However, in the aspect of tool integration, this solution is more to solve the problem of integrating some of the known applications to the clients, rather than to provide a general solution for tool integration.

Finally, Eclipse as a well known Integrated Development Environment (IDE), is a good platform to provide a uniform GUI for different tools. However, it is more suitable to develop a new tool by following it’s standard and reusing it’s common GUI, not for extending the existing tool to be integrated as a whole solution. Also, it is more of a tool to integrate different tools on one machine, and to improve single engineer’s capability, rather than to provide a good integration solution in the team collaboration scenario.

### 2.3 Related Work on Tool Discovery

#### 2.3.1 Current Status on Tool Discovery

Tool adapter discovery is the automated discovery and configuration of the tool adapters in the whole tool chain through minor information given from the user. With tool adapter discovery, the architecture design and deployment of the tool chain can be largely eased. More over, it becomes the reality to reuse the tool adapters from other vendors as part of the tool chain.

However, as we presented before, because the other modern efforts on tool integration are paying more attention to solve a more detail problem, there’re not much

---

\(^9\)http://tomcat.apache.org/
efforts on tool discovery in general. In the following part, we will present the general web service discovery approaches and conclude the different approaches by summarizing the compatibility with different orchestration methods. The limitations on the general web service discoveries are the major reason to develop this specific discovery routing for tool adapters based on OSLC.

2.3.2 Service Discovery and Tool Discovery

Before we present the discovery details, we would clarify the meaning of the web service discovery here. It means the discover of the service, and consume the service by automatical configuration of the running environment. It does nothing with the intelligent discovery of the web services in the research of semantic web, and even the global directory service or search engine service of the web services.

The most well known web service is the one in SOAP, and usually a WSDL (Web Services Description Language) [9][8] document will be included along with the web services. In whatever version, WSDL v1.1 or WSDL v2.0, WSDL is a readme file to document the detail information of the web services provided by the service provider. The consumer, on the other hand, then can consume the web services accordingly.

As a standard in W3C, WSDL is quite mature and is generally accepted. In development with SOAP web services, common programming platforms including .NET and Java that support the consumption of web services all provide facilities to generate stubs to ease the work of interacting with the services provided. Usually, a system library is developed to implement the network interaction details of consuming the web services, and proxy classes that make calls to the library are generated to provide an easy access of the web services for the programmers. In the orchestration of SOAP web services, WSDL is commonly supported in platforms like BPEL (Business Process Execution Language) [1] and SCA (Service Component Architecture) [25]. The discovery of the SOAP web services can be done by providing the address of the WSDL file.

In the other web service world, RESTful web service, things are different. In REST, there’re counterpart as WSDL in SOAP, like WADL [15][14] or WSDL v2.0 [8] presented later to add more support of REST. However, neither way is general accepted. Therefore, in development with RESTful web services, a lot of manual work is required to consume the services provided, not much work can be generated by the tools provided by programming platform. Java has some support of WADL-to-Java generation10, however, most of the service providers nowadays will not provide a WADL description file along with the RESTful web services. In the orchestration with RESTful web services, the current version of BPEL (version 2.0) doesn’t support REST, and the only solutions are to develop adapter services in SOAP or adopt the extension BPEL for REST [27]. However, the former one requires the manual implementation of the adapter services, and the latter one requires manual configuration and doesn’t use WADL either. In SCA, the binding of RESTful web services is possible, however a common Java interface is used to invoke the web services, and no facilities existed

10 http://wadl.java.net/
Compatibility
OSLC Discovery [34]
WSDL 1.1 Discovery [9]
WSDL 2.0 Discovery [8]
WADL Discovery [15]
TIL Discovery

<table>
<thead>
<tr>
<th>Compatibility</th>
<th>OSLC Discovery</th>
<th>WSDL 1.1 Discovery</th>
<th>WSDL 2.0 Discovery</th>
<th>WADL Discovery</th>
<th>TIL Discovery</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPEL v2.0 [1]</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>BPEL for REST [27]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Manual Coding (Java)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>TIL Orchestration</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
</tbody>
</table>

Table 2.1. *Compatibility between Discovery and Orchestration Methods*

to discover and generate the interface automatically. That’s the major reason that in this thesis, we build our own discovery approach only based on the directory services provided by OSLC.

To conclude, we summarize the compatibility between discovery and orchestration methods in Table 2.1.
3. Technical Background

3.1 Introduction

In this chapter, we review the technical background that is related to the thesis work. To better understand the technical background, the technologies are introduced in the group as they are related with each other.

The most important high level concept in the thesis is model-driven architecture (MDA) [29]. With this idea, meta model (or structure definition of the model) of the tool adapter is created and generation rules on the basis of the tool adapter model are compiled. With the standard tool adapter model and the generation tool to generate the skeleton of the tool adapter, the standardization of the tool adapters can be guaranteed. Also, the automated discovery of the tool adapters from other vendors is made possible.

To further ease the work of developing the integration framework, and the work of the final user to develop the integration solution, we take EMF as our modeling basis as its maturity in the modeling and tool support. To better standardize our framework, we follow the becoming industry standard, OSLC. To ease the way of encapsulating the tool adapter, and orchestrating them, we follow SCA as the main architecture of the generated tool chain.

In the following sections, technologies concerning EMF, OSLC, and SCA as mentioned are introduced in detail.

3.2 Eclipse Modeling Framework, EMF

3.2.1 EMF and the Ecore Model

Generally speaking, EMF or Eclipse Modeling Framework, is a guideline of the modeling and a set of code generation facilities that gives the user the power to define a model and operate different transformations on the model through GUI or programming. EMF is also the cornerstone of the Eclipse modeling tool Papyrus\(^1\), where the created UML models are based on EMF model and the model-to-code generation engine of the tool is also based on the facilities provided by EMF.

Ecore model, or sometimes Ecore meta model is acted as the meta model of the models in EMF. Ecore model itself is a EMF model as well, and can be described with itself as its meta model. Figure 3.1 is the simplified structure of Ecore meta model.

\(^{1}\)http://www.eclipse.org/modeling/mdt/?project=papyrus
In this thesis, and also in the team’s research work, all the models are defined based on EMF and its Ecore meta model, including the model of the tool adapter and the TIL model mentioned before. With EMF and the generation facilities provided by and on top of EMF, it becomes possible for us to generate the tool adapter stubs with the rules predefined.

EMF has provided a simple modeling environment, based on which, EMF models can be created and corresponding Java codes can be generated through a few operations in the GUI. The Java classes and interfaces generated then can be used as the data structures, and behave exactly as defined in the model. We can compare EMF model to the UML class diagram despite the much larger description capability in EMF. The models defined in EMF are similar to the models created in the UML Class diagram and the generation process of EMF is equivalent to the generation process of UML Class diagram.

EMF has also provided a lot of facilities to operate on the model pragmatically. When creating the tool adapters, through Ecore code generation, tool adapter model especially the data part is used as data structures in the interactions with the tools and system components. Also, in the automated discovery of OSLC-conform tool adapters, EMF models are created dynamically as the data structure to hold the structural information of the discovered tool model with libraries provided by EMF.

3.2.2 Acceleo

Acceleo [20] is one of the code generation tool provided on top of EMF. More specifically, Acceleo is a pragmatic implementation of the MTL (MOF Model to Text Language) standard [22] by the Object Management Group (OMG). The concept of Model to Text is quite simple, which means the rules are defined on the basis of the understanding of the model structure, and different texts can be generated accordingly.

Acceleo provides a full development environment integrated in Eclipse, where the developers can create model-to-text rules, based on which, codes or other configuration files can be created dynamically by parsing the different content of the model. The Acceleo engine can understand the model structures based on Ecore meta model
or other meta models based on Ecore, therefore it is very simple to develop the rules from the model to the target texts, as no codes needed to parse the structure of the model.

In the thesis, Acceleo is the development platform of the model-to-tool-adapter code generation engine. Rules are predefined, and different codes of the tool adapter can be generated according to the different services defined in the model.

3.3 Open Services for Lifecycle Collaboration, OSLC

3.3.1 OSLC and REST

As introduced in the previous chapter, we follow OSLC to keep abreast of the becoming industry standard and reduce the cost to develop tool adapters for the future tools that support OSLC. Also, by following OSLC, the research team can adopt the designed resources and service interfaces in common scenarios directory, and therefore, the work of designing the integration framework as well as designing tool adapters for the integration solution will be largely eased by reusing the standardized resources that should be provided by the tools.

OSLC core specification defines some general principle of OSLC-conform services. As discussed in Chapter 1, the integration framework concerned in this thesis is web service based, with which data and functionality resources are encapsulated as services. More over, because of the resource based nature of the tool adapters, we choose to adopt the RESTful kind of web services which is also the choice of OSLC. For the tool adapters, to publish the provided service information, a directory service or similar is needed. However, unlike web services in SOAP which usually have a WSDL as the directory service, there’s no common accepted directory service for RESTful web services besides WADL and WSDL 2.0, which is not widely used in practice. Therefore, to facilitate the information exchange between different tools, OSLC defined its own directory service in its core specification. That’s another reason that we adopt OSLC in our integration solution.

As depicted in Figure 3.2, information of services provided by OSLC-conform tools are retrieved through the following steps:

1. ServiceProviderCatalog, the catalog of ServiceProviders as indicated from its name, is the starting point of the query. URIs of ServiceProvider are retrieved by querying the ServiceProviderCatalog resource through its URI and analyzing it.

2. ServiceProvider is the description of the services (or part of the services) provided. Details of the services provided by OSLC are included as the queryCapability and creationFactory in the Service resource inline. ResourceShape or ResourceType may be included as a property of the resource provided, and can be used to understand the structure of the provided resource.

As in RESTful web services, one can get, create, update and delete a resource by executing the HTTP commands GET, POST, PUT, and DELETE. In OSLC,
creationFactory provides the URI to create the new resources by executing the POST command and queryCapability provides the URI to get a list of resource information by executing the GET command. In the listed resources, URLs of a single resource can be obtained, through which the resource can be queried, updated or deleted by executing the GET, PUT and DELETE commands.

3.3.2 OSLC and RDF

Besides the features of OSLC discussed before, OSLC has also indicated one should at least provide a representation in RDF (Resource Description Framework) [17] for the resources provided. In the thesis, we provide RDF and XMI [23] representations for the resources represented in EMF model through a system component called the Converter Service introduced in 5.2.2.2.

As introduced in [17], RDF is a framework for representing information in the Web. Generally speaking, it specifies the way to persist an instance of a special object in a specific XML format.

In RDF, resources are identified with URIs. The RDF consists of expressions as a triple which includes a subject, an object and a predicate. A simple example of this relationship is "Someone (Subject) has a name (predicate) whose value is John (Object)", where the subject or object may be identified with a URI. Also, because of the context of using it in internet environment, the structure of the RDF resources is not strict, which implies that everyone can make any changes to any resources. That’s the reason that OSLC must provide a special resource ResourceShape to describe the exact structure of a specific resource.
3.3.3 Implementation Details: JAX-RS, Jersey, Apache CXF and FreeMarker

In the implementation level, we follow JAX-RS or Java API for RESTful Web Services [28], to create RESTful web services in Java. RESTful services are defined by adding special annotations to the ordinary Java methods, and service bindings are done automatically by the running container according to the deployment configurations. Different libraries including Jersey [26] and Apache CXF [4] are used sequentially in different stages as introduced in Section 4.3.2.

The directory service is implemented in a template-based way where responses are generated by FreeMarker [10] based on the templates of the ServiceProviderCatalog, ServiceProvider and ResourceShapes predefined or pre-generated combined with the specialized information retrieved from the tool adapter model.

In the discovery of OSLC-conform tool adapters, Jersey API (Client) is also used to fetch the responses of a specific URI with a specific HTTP command.

When dealing with RDF models, Jena RDF API [19] is used either to create RDF response from EMF model or to analyze the RDF resource to convert it back to EMF model.

3.4 Service Component Architecture, SCA

3.4.1 SOA and SCA

Service-Oriented Architecture (SOA) defines a set of principles to design the software products as a set of services interacting with each other. SCA, or Service Component Architecture, extends the SOA requirements, and provides an extensive set of features to standardize the way to create software components and the mechanism for describing how those components are working together as an application [7].

As specified in [25] and introduced in [7], SCA components can be built with multiple technologies, it could be Java or other languages using SCA-defined programming models, it could also be other technologies like BPEL. One application could have multiple components that have different technologies, and those components could also run in a single process, in multiple processes or even on multiple machines. In whatever way, the integration of components are described with a common assembly model.

In SCA, a component is defined as the minimum unit of the architecture to provide a set of services, whereas, a composite is defined as several components combined into a larger structure. Composite is only a logic unit, thus different components in one composite can run on different machines. Technically, each composite is described in a separate XML-based configuration file following the Service Component Definition Language (SCDL) and ending in ".composite". Figure 3.3 from [7] depicts the scenario that multiple composites and components are running on multiple machines.
No matter what technologies used, the created component always has the same structure as depicted in Figure 3.4. Typically, one component will implement some logics as one or several services, which are exposed to be invoked by other components. In the other hand, the component may want to interact with other services from alien components. It is done by the invocation of the references of the objects that implements the services provided by other components. The invocation could be the invocation of member function of real objects in Java, or it could be the invocation of web services. Even non-SCA application could interact with SCA components. The interaction scenario between different SCA components and non-SCA applications is depicted in Figure 3.5. The component may also have some properties, which can be read or changed at runtime.

In the thesis, we follow the SCA specification to build the tool chain architecture. The tool adapters as well as the system level components are encapsulated as separate composites, and the interactions between the components are specified in the ".composite" configuration files. In SCA, the relationship between components are specified in loose-coupled configuration file other than hard coding and with Frascati which will

Figure 3.3. SCA example with multiple composites and components running on multiple machines [7]
Figure 3.4. SCA component structure [7]

Figure 3.5. SCA component interaction example [7]
be introduced later, the interactions between the components can even be managed at runtime.

3.4.2 Frascati and Maven

Frascati [30] is one of the several implementations of SCA specification. As summarized in [31], in addition to the features specified by the SCA specification, Frascati provides more support of manageability and configurability expected as an SCA platform. As needed in our research, the whole tool chain can be monitored by Frascati, and the properties of components or component interactions can be managed in runtime.

With the help of Apache Maven [18], a software project management and dependency management tool, Frascati application can be configured to run in different ways easily, e.g.: in FraSCAti Explorer with or without FScript plugin, in FScript Console, or in standalone execution.

With the tools provided by Frascati, a SCA application can be managed in different ways. Figure 3.6 depicts the scenarios to manage an simple helloworld SCA application with FraSCAti Explorer. This helloworld SCA application is included in the examples in Frascati runtime 1.4, and has a very simple architecture depicted in Figure 3.7. The only functionality provided by this SCA application is to print some characters. With FraSCAti Explorer, we can do the following:

• View the information of "Helloworld - pojo" composite (picture 1), "client" and "server" components (picture 2-3), etc.
• Execute the "r" service (picture 4-5).
• Change the value of the "header" property of the "server" component (picture 6).

At runtime, Frascati will connect components as configured in ".composite" configuration files. Proxies will be created according to the different wire type, i.e.: wire via Java objects or via web services. In runtime, similar as the simple example presented before, the administrator can monitor the running state of the service, and can change the connections or configurations (value of the properties) dynamically.
Figure 3.6. A simple Helloworld SCA application running in FraSCAti explorer

Figure 3.7. Architecture of the simple Helloworld SCA application
4. Approach

4.1 Introduction

The work of this thesis starts from an existing Matlab/Simulink Tool Adapter which can provide essential data services in an "OSLC-conform" way. Compared to the expected user scenarios, the development is done iteratively to extend the existing tool adapter to match the expected scenarios. The details of the user scenarios and the iterative development process are introduced later in this chapter.

On the basis of the developed tool adapter, the model and code skeleton of a general tool adapter are generalized, and the generation engine from the tool adapter model to the tool adapter code stubs is developed. The work of the construction platform is tested by creating the second Matlab/Simulink tool adapter from the generated tool adapter code stubs. The new Matlab/Simulink tool adapter is also used as the discovery target in developing the automated discovery tool for unknown OSLC-conform tool adapters. Details of the construction platform as well as the construction process of tool adapters based on it, and the discovery process of the unknown tool adapters conforming to OSLC are presented in the later chapters.

4.2 User Scenarios

4.2.1 Needs in Tool Adapter Construction

In the optimal cases, we would like to create a "template" for all tool adapters. The developer of the tool adapter should be able to provide sufficient information for us through modeling, and our generator should be able to generate as much codes as possible based on the model. The duplicate codes to fulfill the different kinds of standards or specifications should be fully generated, and the developer should only pay attention to the tool specific codes [5].

With EMF, OSLC and SCA, we would like the final developer of the tool adapter first create an EMF model of the tool adapter based on our guidelines, and then our generation engine mainly developed in Acceleo should generate codes and configuration files that deliver the OSLC-based services and encapsulate the tool adapter as a separate composite in SCA. Finally, the developer should add implementations to the tool specific services and test it.
4.2.2 Needs in Automated Tool Adapter Discovery

Because of the standardization of the tool adapters through template-like generation based on the same structure of the tool adapter model, discovery of OSLC-conform tool adapters created by other vendors is made possible.

In the discovery of an ordinary web services, the user would only need to provide minor information such as an URI, and corresponding codes, configuration files, etc, are created by the system, and then can be used or integrated as needed. When adding a new tool adapter, no matter it is developed internally or by the other vendor, it is the same that only one URI is required and the platform then should be able to recognize and configure the services automatically.

4.3 Iterative Development

The work on this thesis has been performed in several iterative steps from the existing Matlab/Simulink tool adapter.

4.3.1 From the Existing Matlab/Simulink Tool Adapter

From the very beginning, there’s a Matlab/Simulink tool adapter available to provide the data services including "Block", "Line" and "Port". The services are exposed as RESTful services, and the responses are presented in a RDF-like way which is partly conforming to OSLC.

After the careful investigating of the existing tool adapter, the following parts that should be refined or extended are discovered:

- In the existing tool adapter, the server container holding the web services is running inside Matlab. That’s because of the special architecture of Matlab with an embedded JRE and the lack of suitable API to invoke Matlab functions. However, this architecture is so complex and so special that it cannot be generalized as an ordinary tool adapter. Also, too much complex operations to manage the embedded server inside Matlab are included. This part should be largely modified to a normal API calling way, as actually a Java RMI based library is available to invoke Matlab codes from outside.

- RDF representation is not exactly following OSLC. This part should be fixed in the future. Also, to fulfill the specification concerning the different HTTP commands, and to deal with scenarios like exceptions in the execution, the current library should be extended.

- Services of control functionalities should be added, like select, startSimulation of Matlab/Simulink.

- Directory services specified in OSLC should be added, including ServiceProvider-Catalog, ServiceProvider and ResourceShape.
4.3.1.1 Improvement: Servlet based Solution

To solve the major problem of the existing tool adapter, a Java API called matlabcontrol\(^1\) is used to interact with Matlab. The library makes use of the Java RMI server running inside Matlab, and can execute the Matlab codes from Java programs outside Matlab. By adopting this API, we change to a servlet based solution in which web services are developed in Jersey, and specialized functionalities are delegated to Matlab codes by invoking the matlabcontrol API. The codes of the tool adapter do not include functionalities like web server startup and management, in the contrary, the services are deployed directly on the Java application server like Tomcat.

This servlet based solution provides a normal case of the tool adapter:

- Interactions with the tools are done through APIs or similar way from outside the tool.
- Only web services are concerned in the tool adapter, no more codes like the management of web servers or service containers included.
- The management of application servers or other containers should be done external to the tool adapter.

4.3.1.2 Improvement: RDF Converter Library

Problems of RDF presentations for EMF objects are fixed according to OSLC specification and RDF specification.

4.3.1.3 Extension: Add Control Services

Control services are implemented as RESTful web services in the same way for the data services, but only invocable through the **POST** command type. The services are defined as *operations* in a special *Class* in the tool adapter model.

4.3.1.4 Extension: Add OSLC Directory Services

OSLC directory services are firstly implemented in the basis of JSP templates as in the example provided by OSLC\(^2\). More specifically, JSP templates for *ServiceProviderCatalog* and *ServiceProvider* are created, and tool adapter related information is passed as page arguments. For *ResourceShape*, because of the great differences of the structures of different resources, it is generated beforehand based on the model.

4.3.2 Migration to Frascati

To integrate the tool adapters in the Frascati platform, after the improvement and extension of the existing tool adapter, the architecture is migrated to Frascati.

\(^1\)A Java API to interact with MATLAB, http://code.google.com/p/matlabcontrol/

Generally speaking the following tasks are done during the migration:

- Architecture of the tool adapter is redesigned. Service annotations and service implementations are separated and libraries like RDF converter are extended and encapsulated as a separate SCA composite included in the system library.
- RESTful services are slightly changed because of the change of implementation library of JAX-RS: from Jersey to Apache CXF.
- Additional components like the ID Resolving Service are introduced.
- Dependencies of the projects are changed to be managed by Maven, which is more convenient and Frascati compatible.
- OSLC Service Providers, ServiceProviderCatalogs and resource shapes are created with FreeMarker as JSP is not naturally supported in Frascati.
5. Construction of OSLC-conform Tool Adapters

5.1 Introduction

As introduced in Section 1.2, in the big picture, the tool chain is constructed as the composition of different tool adapters encapsulated as SCA composites. Here, in this chapter, we mainly discuss the way of constructing an OSLC-conform tool adapter that can be included in the tool chain as a SCA composite.

A tool adapter is used as the bridge between the tool technology space and the tool chain technology space, and an OSLC-conform tool adapter is the tool adapter that helps the tool to provide OSLC-compatible services in the tool chain. Those OSLC-compatible services include:

- Data services: Different types of objects in the tool model are encapsulated as resources in RESTful web services. Operations on those resources are defined and executed through the **GET, POST, PUT, DELETE** commands of HTTP.
- Control services: Other operations provided by the tool are encapsulated as special resources in RESTful web services as well. Operations on those resources are defined and can only be executed by **POST** command of HTTP.
- OSLC services: OSLC directory services and OSLC meta information query services are provided according to the OSLC specification in RESTful web services.

As people did in most of other integration solutions introduced in Chapter 2, a lot of redundant work is needed to construct different tool adapters for different tools. In the context of constructing our OSLC-conform tool adapters encapsulated in SCA, such work includes programming of service interfaces and implementations that follow the SCA specification, programming OSLC-compatible interactions and RDF-based output, etc. The repeated implementation of those redundant is not only a wasting of time, more over it may be error-prone to the final products. The thesis work here, instead of presenting a case to build a new tool adapter directly, we start our work by building the platform to ease the construction process first, and in this basis, only minor work needed to implement the working tool adapter with tool specific functionalities.

The idea of building the platform to ease the construction process is from the model-based approach when designing large software products. For our tool adapter construction platform, it is followed by the similar way as depicted in Figure 5.1. The user of the platform first needs to design the model of the tool adapter following OSLC and our guidelines, and after that, the OSLC-conform tool adapter stubs that can be included in the SCA-based tool chain will be generated. The user then needs to
implement the tool specific codes to fulfill the functionalities designed in the model. After that, the tool adapter is done, and can be used in the tool chain after testing and validation.

![Control flow for OSLC-conform tool adapter design based on the construction platform](image)

Figure 5.1. Control flow for OSLC-conform tool adapter design based on the construction platform

Compared to traditional way, a lot of work is done by the tool adapter construction platform, and the user can pay more attention to implement the tool specific functionalities. Also, by following the model-based approach, and our guidelines when designing the model or implementing the functionalities, the standardization of the tool adapters are guaranteed, based on which, automated discovery process in Chapter 6 is made possible.

5.2 The Tool Adapter Construction Platform

As introduced before, the general idea of the platform is to provide an environment to the user to design the tool adapter models on their own, and based on the model, the tool adapter stubs can be generated by the platform. In this section, we will introduce the designing and implementation of the tool adapter platform in detail. First, we will introduce the guidelines of the tool adapter model design, and then the architecture of a SCA-based tool adapter to be generated as long as the details of each components will be presented. In the last part, the design of the model-to-tool-adapter generation engine will be discussed.

5.2.1 Guidelines of the Tool Adapter Model Design

The model of the tool adapter is used as the generation source in the platform, and is the encapsulation of all data, control and OSLC services provided by the tool adapter to be constructed. To ease the generation work, and to make sure the user of the platform will provide the complete information for the generation process, in this section, we provide a few guidelines for the user to follow:
• The model should be based on EMF and the Ecore model.

• To ease the generation and integration of data services only, the data services and other services are split into two separate models, one called the data model which contains information of the data services only in its root package, and the other called the core model which contains information of the control services and OSLC services in separate sub-package control and oslc.

• For the data services, the different data resources are presented as different classes in the package. The properties of the resource are either defined as attributes of the class if the data type of this property is a built-in type, or references if the data type is referenced to a non built-in type (usually a data resource provided by the tool adapter). To standardize the data model design, an ifest-common model is predesigned, and ProtoObject class should be identified as the parent class of the classes in the data model, therefore the uuid property as the unique ID of the resource and the name property will be always included in the design.

• For the control services, each of the service is defined as an operation under a special class Control under its package. Parameters of the control services are defined as parameters of the operation. However, the parameters already provided in the platform as summarized in Table 5.1 can be exempted in the model design.

• For the OSLC services, to ease the generation work of the OSLC directory services, OSLC specific meta information should be defined as key-value pairs in annotations of corresponding name.

![platform/resource/se.kth.md.generator.frascati/model/ifest_common.ecore](attachment://platform/resource/se.kth.md.generator.frascati/model/ifest_common.ecore)

```
platform/resource/se.kth.md.generator.frascati/model/ifest_common.ecore
```

![Identity](attachment://Identity)

```
Identity
- uuid: EString
- name: EString
- ProtoObject -- Identity
```

Figure 5.2. Design of the ifest-common model

Please note, as the limitation of the presentation capability in EMF, the additional parameters of the data services cannot be represented in the model. The user of the platform needs to manually process those additional parameters in the implementation codes if applicable. In the future, additional guidelines of modeling additional parameters for data services can be specified.

5.2.2 Architecture of the Tool Adapter to be Generated

The general idea of the architecture design is to design a SCA-based architecture, where components of the tool adapter as well as the the system components encapsulated as SCA composites are connected with wires defined in ".composite" file.
One important consideration of the architecture design is to split the codes fully generated or to be implemented manually into different components to ease the implementation work by achieving separation of concerns. Thus, the tool adapter itself is designed to be split into two parts, one fully generated external part, which mainly contains RESTful annotations and essential implementations to provide the RESTful services to the external world as defined in the model, and one internal part that needs to be implemented to connect the third-party tools.

Also, the platform provides some system level components that are used in the tool adapter. Those components are connected with wires in the configuration file.

In Figure 5.3, the architecture of a typical tool adapter to be generated is presented by the composite diagram of the Matlab/Simulink Tool Adapter. As a typical tool adapter, the adapter itself contains two parts, the external part or SimulinkExternal in the diagram and the internal part or SimulinkInternal in the diagram. The system level components ConverterService and IDResolvingService are encapsulated as separate composites and are connected to the tool adapter.

![Figure 5.3. Composite Diagram of the Matlab/Simulink Tool Adapter](image)

### 5.2.2.1 Tool Adapter Components

Each tool adapter contains two parts, external and internal. Each part includes a Java interface as the Service in SCA, and a Java class as the implementation. The two parts are connected by a wire in SCA. For each tool adapter, a composite file is also included to define the configurations to integrate the components of the tool adapter and the system level components together.

Please note, as the feature in Frascati, the connections can be modified dynamically to other ways. Also, in real deployment, the different components, for example the two parts of the tool adapter, can be deployed in different machines if needed.
Tool Adapter External

The external part of the tool adapter is fully generated on the basis of the tool adapter model designed following our guidelines. The generated component is strictly following OSLC specification and other guidelines we defined.

More specifically, this component includes the following part:

- RESTful service declarations and implementations of the data and control integration functionalities: all the actual implementations and parameter handling are delegated to the internal component of the tool adapter.
- Full implementation of OSLC directory services, including ServiceProviderCatalog, ServiceProvider and ResourceShape services of the data resources. For the implementation details, ServiceProviderCatalog and ServiceProvider services are on the basis of the templates predefined combined with tool specific information extracted from oslc sub-package of the model, and ResourceShape services are directly provided on the ResourceShape templates generated from the model.
- SCA configurations to bind the service to a specific address and wire the helper instance to the internal component.

For the OSLC services, the implementations of the ServiceProviderCatalog and ServiceProvider are based on the following strategies:

- A ServiceProvider lists a category of the data or control services provided by the tool adapter, no ServiceProvider contains both data and control services.
- When representing data services, the creation URI of the resource is in the inlined resource oslc:CreationFactory and the listing URI of the resources is in the inlined resource oslc:QueryCapability. One data resource will have one oslc:CreationFactory and one oslc:QueryCapability.
- When representing control services, the invocation URI of the resource is in the inlined resource oslc:CreationFactory, and no oslc:QueryCapability resource existed.

Please note, because of the lack of control resources support in OSLC, here we make a strong assumption to place the control services properly and distinguish the control services from data services. Also, currently, the OSLC specification does not provide any support on the directory services of the parameters for both the data and control services. In the future, the standard of the directory service for parameters should be established by the OSLC community.

Tool Adapter Internal

This part, contrary to the external component, is required for the user of the platform to fill in the tool adapter specific codes.

To ease the work of the developer of tool adapters and to make sure the developer fills in the codes in the right way, a general framework is predefined and included with examples in the the generated stubs. The work that the developer needs to do includes the handling of parameters from the web service calls, the implementation of interactions of the tools with the tool API, etc. Those parts are all marked as TODO.
in the protected region where custom modification will remain after the re-generation. A wrapper for the parameters from the web service calls and another wrapper for the response from the tool are provided, which the user must use to interact with the predefined framework.

This part also links to the system components to provide the converter service and the ID resolving service, therefore SCA configurations are generated to link the right instances to the system level components.

In the current demo, to support the special functionalities of the tool, several parameters as summarized in Table 5.1 are included by the platform which the user of the platform is suggested to follow and make use of.

**Table 5.1. Parameters predefined in the construction platform**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model_name</td>
<td>String</td>
<td>To support the ID resolving service, if the tool is project or model based, this parameter must be included in the service call or default value predefined should be used in the implementation, and the parameter must be extracted and then set as the <code>ServiceParam</code> of the ID resolving service.</td>
</tr>
<tr>
<td>subtree_levels</td>
<td>Integer</td>
<td>It is an optional parameter to enable the multiple subtree-levels access responses. Refer to Section 5.2.2.2 for more information.</td>
</tr>
<tr>
<td>converter</td>
<td>String</td>
<td>It is an optional parameter only when the automated-selection output service is selected as the current converter service. XMI converter service will be called if the value is &quot;xmi&quot;, RDF converter service will be called otherwise. Refer to Section 5.2.2.2 for more information.</td>
</tr>
</tbody>
</table>

### 5.2.2.2 System Level Components

Two system level components are included in the current platform: the ID resolving service and the converter service. Both the services are encapsulated as SCA composites, and included in the platform libraries.

**The ID Resolving Service**

The ID resolving service mainly manages all the different kinds of IDs of objects in different tools. A system ID that looks no difference in different tools is always used outside the tool to hide the implementation details. The service is implemented independent of the tool adapters, the same codes will be used for all tool adapters, although different instances will be created if running on multiple machines.

This service mainly provides a resolving service (resolving from system ID to tool ID) and a backward resolving service (resolving from tool ID to system ID). Usually,
it is the responsibility of user of the platform to call the resolving service to convert the system IDs in the web service parameters to the tool IDs used inside the tool, while for the response from the tool, in normal cases (if those IDs are named as `uuid` and used as the key of the object), the specific IDs are automatically converted to system IDs by the converter service.

To use this service, pre-registration of the initialization method is required which will be automatically called when the current tool (with tool project/model name if available) is not recorded in the database. This initialization method should return all the tool IDs in the tool as a list, and the service will bind a generated uuid as the system ID for each of the tool ID. When in the creation or deletion of the resources, corresponding calls should be made to the service to create new ID mappings or delete old mappings.

For the current implementation of this service, an embedded SQLite\(^1\) database is included to store the ID pairs (tool ID and system ID pair with tool adapter name and project name if applicable) of each tools. A more sophisticated solution can be made in complex scenarios.

Please note, this service can be disabled by not linking the corresponding reference in the internal component of the tool adapter to the IdManager implemented in the library.

**The Converter Service**

The converter service is the service to output the resources from the tool in a specific standard way. For the current platform library, a RDF-based output service, a XMI-based output service and an auto-select service based on a specific parameter are provided. The converter service is also a tool adapter independent component, but different tool adapters will use a separate instance of the service. With Frascati injecting tools, the current response type of the web services can be modified by changing the wires of the components on the fly.

In general cases, the output style of the resources is followed by the examples from OSLC which will usually presents references of the resources as external URIs (or IDs when in XMI-based output style). However, for RDF-based and XMI-based services, the output styles are of tiny differences when output a list of the resources or output the details of a resource. Those differences are summarized in Table 5.2.

For the auto-select service, a special parameter called `converter` is extracted. XMI converter service is called if the value of the parameter is "xmi", RDF converter service is called otherwise.

This converter service also supports a feature called "multiple subtree-level access" through a parameter `subtree_levels`. If this parameter is not provided, or the value of the parameter is 0, the general way of the output as described before and summarized in Table 5.2 will be used, otherwise, the external URIs or (IDs) will be replaced as inline resources for whatever output service type is. With the increment of the value, the inlinement of the resources will be to a larger depth.

\(^1\)http://www.sqlite.org/
Table 5.2. Different output styles of XMI-based and RDF-based output services

<table>
<thead>
<tr>
<th>Service Type</th>
<th>Output Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDF-based</td>
<td>List of Resources</td>
<td>Resources are listed with external URIs only</td>
</tr>
<tr>
<td>XMI-based</td>
<td>List of Resources</td>
<td>Resources are listed with IDs and values of other attributes, no references included</td>
</tr>
<tr>
<td>RDF-based</td>
<td>Details of a Resource</td>
<td>Values of non-empty attributes are included, non-empty references are included as external URIs only</td>
</tr>
<tr>
<td>XMI-based</td>
<td>Details of a Resource</td>
<td>Values of non-empty attributes are included, non-empty references are included with IDs and values of other attributes, no references of references included</td>
</tr>
</tbody>
</table>

With the converter service, responses can be automatically converted to the right type with the considerations of different operation statuses and different categories of results. Examples are special response when the resource is not found or execution exception is thrown.

The converter service may be initialized with an ID resolving service. In this case, before the conversion of the response, data in all key fields (named as `uuid`) will be replaced by the system ID from the backward resolving service.

Please note, in the implementation of the converter service, we assume that the key of the resource is named by `uuid`. To meet this assumption, please follow the guidelines in designing the model.

5.2.3 Generation of the Tool Adapter from a Model

This part will discuss the generation from the model to the tool adapter. The generation here is a Model-to-text transformation, and is separated as two parts:

For the first part, it is the Java implementation generation of the model we created. This part, the transformation process is already provided by EMF. In EMF, based on the model, the relevant data types represented as classes can be generated to Java source codes. The generated classes can be used when interacting with the tools as data structures and our system level components will only operate on the objects of the generated classes.

After the design of the model based on Ecore, a generation model can be created and then the codes of the model can be generated through a few clicks on the GUI. However, to integrate all the generation work together, a separate utility for this purpose is developed to be included in the generation engine.

For the second part, it is the generation of tool adapter stubs based on the model we created. Different to the generation of the model, here we need to develop our
own rules for the *Model-to-text* transformation process. Based on Acceleo, rules are defined in the following parts:

- Java codes for both the external and internal component of the tool adapter. In general, the external component is fully generated, however, only partial of the codes are generated for the internal component, and more needs to be filled by the user of the platform. For EMF based tools, more codes can be generated as a common implementation of tool specific codes exist.
- SCA composite file for the tool adapter with wires between the corresponding components.
- Freemarker templates to be used in the OSLC directory service implementations in the external component of the tool adapter. With Freemarker, templates for the general structures of the OSLC defined resources can be created beforehand, and with the information fetch at runtime, specialized output can be created easily.
- POM file of the tool adapter which is used to configure and run SCA middleware through an easy way.
- Other setting files including an optional client setting file which can be used in the deployment or by the user and an optional *log4j* configuration file.

More specifically, in the internal component, the interaction details in the implementation class should be implemented by the developer. Those details are marked with **TODO** in the protected region noted between *Start of user code XXX* and *End of user code XXX*. The user can also include user specific codes in the protected region. In summary, developer should implement the following parts:

- Register the *ID Resolving Service* idManager with instance of class that implements IInitIdService (to support the initialization of the *ID Resolving Service*).
- Register XMI parser and user meta model if needed, e.g.: if the user needs to read or write resources based on those models.
- Implement getNS of different service types (data or control), the result will be included in the response.
- Complete `processRequest` method with the service name and parameters included: process the event object generated and set the answer as part of the event. It is suggested that the developer creates user specific classes in other packages and calls those classes here.
- Handling of parameters in `prepare` method and all other help methods declared for the services provided. Parameters can be validated and added with/without ID resolution.

Also, in the generated configuration file of the SCA based tool adapter, deployment configurations like the binding URIs of the services should be modified accordingly.

Please **note**, the generation engine provided in the platform is actually based on the TIL model as discussed in Section 1.2 to provide the generation capability for the whole tool chain. In the big picture, each tool adapter will have its own external and internal components, SCA composite file for its own configuration and OSLC based resource shape templates, but there will be only one copy of templates to provide different ServiceProviderCatalog and ServiceProvider services for different tool adapters together with the tool adapter meta information retrieved from the model, and
also there will be only one POM file, one setting file and one \textit{log4j} configuration file for the whole tool chain.

5.3 Construction of Tool Adapters on the Platform

In this section, we present the process to construct a tool adapter based on the platform as introduced before. A case study of constructing a Matlab/Simulink tool adapter on the platform is provided through the process.

5.3.1 Model the Data and Control Integration

The first step of the construction of a tool adapter is to design the tool adapter model following the guidelines discussed in Section 5.2.1.

Before that, the data and control integration services to be provided are identified through the analysis of user requirements. For the simplification of the example, we assume we will provide the following services of the Matlab/Simulink tool as summarized in Table 5.3 and Table 5.4. Please \textbf{note}, those services are not complete for the tool, and the encapsulation of the services may not be 100\% accurate for the tool.

\textbf{Table 5.3. Data Services to be provided in Matlab/Simulink}

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block</td>
<td>ID, name, block type, parent block, sub-blocks, containing ports, containing lines</td>
</tr>
<tr>
<td>Connection</td>
<td>ID, name, parent block, source port, destination port</td>
</tr>
<tr>
<td>Port</td>
<td>ID, name, parent block, port type, port number</td>
</tr>
</tbody>
</table>

\textbf{Table 5.4. Control Services to be provided in Matlab/Simulink}

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load model</td>
<td>Model name</td>
</tr>
<tr>
<td>Save model</td>
<td>Model name</td>
</tr>
<tr>
<td>Open model</td>
<td>Model name</td>
</tr>
<tr>
<td>Close model</td>
<td>Model name</td>
</tr>
<tr>
<td>Start simulation</td>
<td>Model name</td>
</tr>
<tr>
<td>Select component</td>
<td>Model name, ID of the component, Flag to select or de-select</td>
</tr>
<tr>
<td>Export</td>
<td>Name of the model to be exported</td>
</tr>
<tr>
<td>Import</td>
<td>Name of the new model to be imported, Content of the model in XMI</td>
</tr>
</tbody>
</table>

Also, there is some meta information to be included in the model to implement the OSLC directory services.

32
Following the guidelines, and take the consideration of parameters predefined in the platform, we design the data model and the core model of Matlab/Simulink as in Figure 5.4 and Figure 5.5.

5.3.2 Generate the Tool Adapter Stubs

By executing the Ecore code generation process provided by EMF, a Java implementation of the tool adapter model is generated. Those classes are used as data structures when working with the tool adapters. For our example of the Matlab/Simulink tool adapter, the Java source files generated are presented as in Figure 5.6.

By executing the model-to-tool-adapter generation process, we will get the whole tool adapter stubs as introduced before. For our example of the Matlab/Simulink tool adapter, the files generated are presented as in Figure 5.7.

For a complete reference, the most important part of the generated codes are included in Appendix A.

Please note, here presents the generation result of the TIL model containing the model of Matlab/Simulink adapter, therefore files for the tool chain will also be generated.

5.3.3 Implementation of the Tool Adapter

In this part, the developer of the tool adapter should fill in the blanks in the generated results as summarized in Section 5.2.3. Generally speaking, there are three categories of the tasks should be manually involved by the developer, the settings of the initializations parameter of the system components provided by the framework, the parameter handling of the web service calls because of the limitation in parameter modelings and the manual implementation of the tool related functionalities.

Compared to other tasks, the manual implementation of tool related functionalities takes up most of the efforts. In the following part, the design details of the manual implementation part of the Matlab/Simulink adapter are presented.

As introduced in Section 4.3, this Matlab/Simulink adapter is built on an existing tool adapter. Architecture of the tool adapter is refined to fit the general development process in the framework through a third-party Java RMI based library matlabcontrol, and implementations of the control functionalities are added.

More specifically, as suggested in Section 5.2.3, an external class MatlabProxy-Wrapper is created to delegate all the related requests to the Matlab codes through matlabcontrol. In Matlab, the requests are analyzed and routed to the right class, and the final results are returned through data structures from EMF model generation. The manual part of the architecture of Matlab/Simulink tool adapter is depicted in Figure 5.8.
Figure 5.4. Design of the data model of Matlab/Simulink Tool Adapter

Figure 5.5. Design of the core model of Matlab/Simulink Tool Adapter
Figure 5.6. Java implementation generated of the data and core model of Matlab/Simulink tool adapter

Figure 5.7. Tool adapter stubs generated based on the data and core model of Matlab/Simulink tool adapter
Figure 5.8. Architecture of Matlab/Simulink Tool Adapter (Manual Part)
5.3.4 Testing and the Result

After the implementation of the tool adapter (or tool adapters), the tool chain can be tested simply by the Maven command `mvn clean install -P run`.

Below we present the testing result of the implemented Matlab/Simulink adapter with the designed model `york` as displayed in Figure 5.9 (sub-blocks are also presented in hierarchies).

Because of the limitation of the space in the thesis report, we only include the response of `GET` a list of `Block` resource and `GET` the details of one `Block` resource with sub-tree access level 1 in RDF and XMI output type. For the OSLC directory services and some other examples, please refer to the step by step analysis of the discovery algorithm with the example of this tool adapter in Section 6.2.2.

`GET` a list of `Block` resource with sub-tree access level 1 in RDF with URI http://localhost:8080/Simulink/block?subtree_levels=1&converter=rdf

```xml
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:oslcsimulinkdata="http://md.kth.se/oslc/simulink/1.0/data#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
    <rdf:Description rdf:about="http://localhost:8080/Simulink/block">
        <oslcsimulinkdata:Block rdf:about="http://localhost:8080/Simulink/block/1fc429e1-a0e6-4316-b5bb-8a1f2630b87c">
            <oslcsimulinkdata:uuid>1fc429e1-a0e6-4316-b5bb-8a1f2630b87c</oslcsimulinkdata:uuid>
            <oslcsimulinkdata:type> Gain</oslcsimulinkdata:type>
            <oslcsimulinkdata:name> F3</oslcsimulinkdata:name>
            <oslcsimulinkdata:children rdf:resource="http://localhost:8080/Simulink/block/6fe4af76-4c77-9f0f-ae76f1755754"/>
            <oslcsimulinkdata:lines rdf:resource="http://localhost:8080/Simulink/connection/80940741-8d6e-46cc-97a3-0d1c7c0c18bf"/>
        </oslcsimulinkdata:Block>
    </rdf:Description>
</rdf:RDF>
```
Figure 5.9. Screen shot of Matlab/Simulink model york
GET a list of Block resource with sub-tree access level 1 in XMI with URI
http://localhost:8080/Simulink/block?subtree_levels=1&converter=xmi
GET the details of one Block resource with sub-tree access level 1 in RDF with URI http://localhost:8080/Simulink/block/272fd601-9b74-4726-a4e3-22a57e451873?subtree_levels=1&converter=rdf
GET the details of one Block resource with sub-tree access level 1 in XMI with URI
http://localhost:8080/Simulink/block/272fd601-9b74-4726-a4e3-22a57e451873?
subtree_levels=1&converter=xmi
5.4 Limitations and Assumptions

Here, we summarize the limitations and assumptions made of the current tool adapter construction platform and/or the current Matlab/Simulink adapter.
<table>
<thead>
<tr>
<th>Relevant Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model related</td>
<td>Additional parameters of the data services cannot be represented in the model, thus cannot be generated in the tool adapter stubs.</td>
</tr>
<tr>
<td>OSLC related</td>
<td>No directory services available in OSLC for parameters in data and control services.</td>
</tr>
<tr>
<td>OSLC related</td>
<td>A ServiceProvider lists a category of the data or control services provided by the tool adapter, no ServiceProvider contains both data and control services. When representing data services, the creation URI of the resource is in the inlined resource oslc:CreationFactory and the listing URI of the resources is in the inlined resource oslc:QueryCapability. One data resource will have one oslc:CreationFactory and one oslc:QueryCapability. When representing control services, the invocation URI of the resource is in the inlined resource oslc:CreationFactory, and no oslc:QueryCapability resource existed.</td>
</tr>
<tr>
<td>SCA configurations</td>
<td>Components are linked with wires directly in the default generated configurations.</td>
</tr>
<tr>
<td>SCA configurations</td>
<td>0.0.0.0 can't be used in the binding address of the RESTful web services in the SCA configuration, otherwise the OSLC directory services may not work properly.</td>
</tr>
<tr>
<td>ID Resolving Service</td>
<td>Only one instance of ID resolving service in one execution unit, ID pairs are stored in the local database. Service is not suggested to serve globally.</td>
</tr>
<tr>
<td>Converter Service</td>
<td>The uuid is assumed as the only key of the object.</td>
</tr>
<tr>
<td>Converter Service</td>
<td>Tiny differences in RDF or XMI representations exist as summarized in Table 5.2.</td>
</tr>
<tr>
<td>Converter Service</td>
<td>We use State 500 for exception with exception message as the response, State 201 for created object with a link in the location and created object content as response (according to REST), State 200 for normal response and State 404 for resources not found.</td>
</tr>
<tr>
<td>Relevant Part</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Generated tool adapter stubs</td>
<td><em>Serializable</em> interface is added to the generated models manually, as the objects are transmitted through Java RMI.</td>
</tr>
<tr>
<td>Third-party library</td>
<td>A third-party library is used to invoke Matlab codes from Java, this library however is not stable as it is created by hacking the Java RMI server running inside Matlab.</td>
</tr>
<tr>
<td>Matlab codes</td>
<td>The ID (through TAG) of a connection without the name is not persistent, however the adapter makes sure the name should be provided when creating a new connection.</td>
</tr>
</tbody>
</table>
6. Automated Discovery of Unknown Tool Adapters Conforming to OSLC

6.1 Introduction

To construct a tool chain, it is not possible to build tool adapters for all the tools used in the solution on one’s own. More practically, adapters are constructed by different vendors, shared in creating tool integration solutions. To accomplish this aim, the tool adapters must be following the same standard.

In this chapter, we explain how an unknown tool adapter conforming to OSLC and built on the platform we describe in Section 5.2 can be discovered and then configured automatically to be included in the tool chain.

In the OSLC standard [34], the start point of the services provided is the URI of ServiceProviderCatalog, which gives the same start point for the automatic discovery process. With this URI, we aim to understand the services provided by the unknown adapter, and try to generate the stubs needed to consume the RESTful services. Therefore, the automated discovery of unknown tool adapters is quite similar to the discovery process of a SOAP web service through WSDL. The only difference is that to discover an unknown tool adapter is a more concrete problem compared to the discovery of a general web service. Also, because of the simplified grammar that the RESTful web service is based on, we are unable to get useful information through directory services similar to WSDL, the only possible way to get needed information is to follow the OSLC specification and query the information through OSLC directory services.

The general idea is, by parsing the content of ServiceProviderCatalog, and then following the content of ServiceProvider, ResourceShapes of the resources, etc, we can create a model for the discovered tool. With this model, in the similar way as we do to generate tool adapter stubs to construct a tool adapter based on the construction platform in Section 5.3.2, we can generate discovered tool adapter stubs which can be used to bind to address of the RESTful services. This discovered tool adapter stubs will be encapsulated in SCA component and can be included into the SCA based tool chain easily. In this way, the unknown tool adapter can be automatically discovered and configured, and no more manual configurations needed in the integration process.

As described in the user scenario in Section 4.2.2, in the future, the user who wants to include an unknown tool adapter to the tool chain will only need to provide an address of the start point (the URI of the ServiceProviderCatalog), thereafter, the discovered tool adapter stubs are generated and configured to be included in the tool adapter automatically.
In this chapter, just like the previous chapter, details of the discovery and configuration processes are presented with an example of discovering an unknown tool adapter. To simplify the thesis project, and also because of the unavailability of other tool adapters conforming to OSLC and built on our platform, we use the previous constructed Matlab/Simulink Adapter as the discovery target.

6.2 Approach

6.2.1 Model Design of the Discovered Tool Adapter

Similar to the tool adapter model in the construction process, a discovered tool adapter model is used to encapsulate the data, control and OSLC services provided by the tool adapter. The difference is that the information is obtained through the discovery process, and is only used to generate the stubs we need for consuming the services.

In other words, the model must obey the following rules:

- It only includes information that can be obtained through OSLC services, no others.
- It includes at least the information needed to generate the stubs to consume the services.

With the previous experience of designing EMF based tool adapter models, and rules from the special context of the model here, we use the following strategies to design the model:

- To facilitate the integration steps, we follow the same separate model guideline in the construction process discussed in Section 5.2.1, which is to separate out the data part as a data model, and all the other parts as a core model.
- We also follow the same guidelines as discussed in Section 5.2.1 to separate different types of services in different packages or sub-packages. However, the information in the model is directly from the discovery process, and we don’t guarantee the generated model will be designed in the same way as we do in the construction process, e.g.: classes in data part are not inherited from Prototype in the ifest-common model as before.
- In addition to the information to be provided when designing the model for the construction process as discussed in Section 5.2.1, here the OSLC related information is obtained and recorded as annotations to the items (could be classes or
6.2.2 The Discovery Algorithm

In this section, we present the "discovery" in narrow sense, which is the algorithm to discover the unknown tool adapter, and save the information needed as an EMF tool meta model. The general process of this discovery algorithm includes:

- The start point of the service ServiceProviderCatalog defined in OSLC specification provides all the service providers of this tool adapter. Following the links of each ServiceProvider, we will see a list of provided service which may be control or data service provided by the tool.
- For the control service, we can easily find a list of services including the URI we need to invoke the service, and other information needed like service title and label as defined in OSLC specification.
- For the data service, we can easily find a list of data resources of the tool adapter with their creation URI and query URI. However for the details of the resources, like the attributes included and references to others, we may either obtain from the ResourceShape or response details of a set of objects in the same type.

In Figure 6.2, we present the detailed algorithm of the discovery process.

![Figure 6.2. Detail algorithm of the automated discovery process](image)

For a more clear understanding of the discovery algorithm, we summarize the details of each operation with an example. In the demo for the service discovery, Jersey Client API is used to fetch the response of the RESTful web services, and Jena is used to analyze the RDF-based response as introduced in 3.3.3.
**Step 1: Find ServiceProviders: Parse ServiceProviderCatalog**

By parsing the response of **GET ServiceProviderCatalog** with corresponding URI, we can identify a set of `oslc:ServiceProvider` resources, each of which represents a category of the services provided by this tool adapter. Besides, OSLC related information concerning of the `ServiceProviderCatalog` is extracted and saved.

For each `oslc:ServiceProvider` resource, goto Step 2 for further processing.

In our example, by **GET ServiceProviderCatalog** with the URI http://localhost:8080/Simulink/oslc/service_provider_catalog, we can obtain the following response:

```xml
<rdf:RDF xmlns:oslc="http://open-services.net/ns/core#"
         xmlns:dcterms="http://purl.org/dc/terms/"
         xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <oslc:ServiceProviderCatalog
    rdf:about="http://localhost:8080/Simulink/oslc/service_provider_catalog">
    <dcterms:title>Matlab / Simulink OSLC Service Provider Catalog</dcterms:title>
    <dcterms:description>Enables navigation to Service Provider for each Data / Control integration functionalities of Tool Matlab / Simulink.</dcterms:description>
    <oslc:ServiceProvider rdf:about="http://localhost:8080/Simulink/oslc/service_provider/data">
      <dcterms:title>OSLC Simulink OSLC Service Provider - Data</dcterms:title>
    </oslc:ServiceProvider>
    <oslc:ServiceProvider rdf:about="http://localhost:8080/Simulink/oslc/service_provider/control">
      <dcterms:title>OSLC Simulink OSLC Service Provider - Control</dcterms:title>
    </oslc:ServiceProvider>
  </oslc:ServiceProviderCatalog>
</rdf:RDF>
```

By parsing the content we can identify two `oslc:ServiceProvider` resources with URI http://localhost:8080/Simulink/oslc/service_provider/data and http://localhost:8080/Simulink/oslc/service_provider/control (line 15 and 20) which will be further processed in Step 2. Also, we get the `dcterms:title` and `dcterms:description` properties of the current `ServiceProviderCatalog` resource which can be used in the future.
Step 2: Data or Control Service?

Before this step, we need to obtain the response of oslc:ServiceProvider by GET the corresponding URI provided.

In our example, by GET oslc:ServiceProvider with the URI http://localhost:8080/Simulink/oslc/service_provider/data, we can obtain the following response:

```xml
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dcterms="http://purl.org/dc/terms/"
  xmlns:oslc="http://open-services.net/ns/core#">
  <oslc:ServiceProvider rdf:about="http://localhost:8080/Simulink/oslc/service_provider/data">
    <dcterms:title>
      OSCL Simulink OSLC Service Provider - Data
    </dcterms:title>
    <dcterms:description>
      Provide Data integration functionalities of Tool Matlab/Simulink, e.g. List, Create, Get, Update,
      Delete a Model, Connection or Port.
    </dcterms:description>
    <oslc:service>
      <oslc:CreationFactory>
        <dcterms:title>Location for creation of Block Entries</dcterms:title>
        <oslc:label>Block</oslc:label>
        <oslc:resourceShape rdf:resource="http://localhost:8080/Simulink/shapes/Block"/>
        <oslc:resourceType rdf:resource="http://md.kth.se/oslc/simulink/1.0/data#Block"/>
      </oslc:CreationFactory>
      <oslc:CreationFactory>
        <dcterms:title>Location for creation of Connection Entries</dcterms:title>
        <oslc:label>Connection</oslc:label>
        <oslc:resourceType rdf:resource="http://md.kth.se/oslc/simulink/1.0/data#Connection"/>
      </oslc:CreationFactory>
    </oslc:service>
  </oslc:ServiceProvider>
</rdf:RDF>
```
By **GET oslc:ServiceProvider** with the URI http://localhost:8080/Simulink/oslc/service_provider/control, we can obtain the following response (partial):

```xml
<rdf:RDF xmlns:oslc="http://open-services.net/ns/core#"
         xmlns:dc="http://purl.org/dc/terms/"
  <oslc:ServiceProvider rdf:about="http://localhost:8080/Simulink/oslc/service_provider/control">
    <dc:title>
      OSLC Simulink OSLC Service Provider – Control
    </dc:title>
    <dc:description>
      
    </dc:description>
  </oslc:ServiceProvider>
</rdf:RDF>
```
Here, we follow the strong assumptions made of OSLC directory services in Section 5.2.2.1 that data and control services are encapsulated in separate oslc:ServiceProvider resources, for the control ServiceProvider, only inlined oslc:CreationFactory resources existed, whereas, for the data ServiceProvider, both oslc:CreationFactory and oslc:QueryCapability resources existed. Therefore we check the service type by checking if the current oslc:ServiceProvider contains any oslc:QueryCapability resources.

In our example, the oslc:ServiceProvider with the URI http://localhost:8080/Simulink/oslc/service_provider/data is a data ServiceProvider, whereas, the oslc:ServiceProvider with the URI http://localhost:8080/Simulink/oslc/service_provider/control is a control ServiceProvider.
After the identification the correct type of ServiceProvider, for data ServiceProvider goto Step 3 for further processing, whereas, for control ServiceProvider goto Step 6 for further processing.

Step 3: Find Data Resources

Each inlined oslc:CreationFactory or oslc:QueryCapability property represents one data resource, however for the name of this data resource we have to assume it is from the URI of the inner property oslc:resourceShape, oslc:resourceType, oslc:creation or oslc:queryBase. With the listed sequence, we anticipate the resource name by extracting the last word of the URI. Here, we use this simplest way to anticipate the correct name, which is to select from the first URI in the listed sequence if existed. For a more sophisticated solution, possible names should be recorded, and voted during the next steps when building the references between them. The user should also be given a chance to select the ideal name.

In our example, according to the response of oslc:ServiceProvider with the URI http://localhost:8080/Simulink/oslc/service_provider/data, we can identify three inlined oslc:CreationFactory (see line 17, 26, 35) and three inlined oslc:QueryCapability (see line 44, 51, 58), from which we can find three data resources, or classes in the model, they are Block, Connection and Port. Take the one oslc:CreationFactory with label Block for example, we can anticipate the name of the data resource as Block by extracting the last word of the URI http://localhost:8080/Simulink/shapes/Block of the property oslc:resourceShape.

For each data resource, we need to check if the URI of oslc:ResourceShape resource is given in the response. If it is provided, details of this data resource can be constructed by parsing the oslc:ResourceShape given, otherwise, we need to query one or more specific objects in that data type to anticipate the structure of the resource. Follow Step 4 if the URI of oslc:ResourceShape is given and Step 5 otherwise.

Please note, for the simplification implementation of the demo, we assume the annotated oslc:ResourceShape resource here is given by providing an external URI, not directly inlined in the current response. Also, in the current demo, the discovery of the additional parameters for the data services is not available as no suitable directory service for parameters defined in OSLC.

Step 4: Construct Data Resource Structures: Parse ResourceShape

Properties of the current data resource are analyzed and inserted to the corresponding class as attributes or references. More specifically for properties with the built-in data type like string, int, etc, an attribute is added to the class, otherwise, by analyzing the URI of the given oslc:valueShape or oslc:valueType property, with the same way in Step 3, we anticipate the referenced data type in the same assumption and add corresponding reference to the current class. Numerical attribute of the attribute or reference is also specified by checking oslc:occurs property given. After the construction of the data resource, the model is then updated.
In our example, take the `oslc:ResourceShape` of the `Block` resource with URI `http://localhost:8080/Simulink/shapes/Block` for example, we can obtain the following response:

```xml
<!— generated by the OSLC Adapter Generator for Ecore, Copyright 2011 Matthias Biehl, KTH, http://www.md.kth.se/~biehl/oslccore —>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dcterm="http://purl.org/dc/terms/"
  xmlns:foaf="http://xmlns.com/foaf/0.1/"
  xmlns:oslc="http://open-services.net/ns/core#">
  <oslc:ResourceShape rdf:about="http://localhost:8080/Simulink/shapes/Block">
    <dcterm:title>Block Shape</dcterm:title>
    <oslc:describes rdf:resource="http://md.kth.se/oslc/simulink/1.0/data#Block" />
    <oslc:property>
      <oslc:Property>
        <oslc:name>uuid</oslc:name>
        <oslc:propertyDefinition rdf:resource="http://purl.org/dc/terms/content" />
        <oslc:valueType rdf:resource="http://www.w3.org/2001/XMLSchema#string" />
        <oslc:occurs rdf:resource="http://open-services.net/ns/core#Zero-or-one" />
      </oslc:Property>
    </oslc:property>
    <oslc:property>
      <oslc:Property>
        <oslc:name>name</oslc:name>
        <oslc:propertyDefinition rdf:resource="http://purl.org/dc/terms/content" />
        <oslc:valueType rdf:resource="http://www.w3.org/2001/XMLSchema#string" />
        <oslc:occurs rdf:resource="http://open-services.net/ns/core#Zero-or-one" />
      </oslc:Property>
    </oslc:property>
    <oslc:property>
      <oslc:Property>
        <oslc:name>type</oslc:name>
        <oslc:propertyDefinition rdf:resource="http://purl.org/dc/terms/content" />
        <oslc:valueType rdf:resource="http://www.w3.org/2001/XMLSchema#string" />
        <oslc:occurs rdf:resource="http://open-services.net/ns/core#Zero-or-one" />
      </oslc:Property>
    </oslc:property>
    <oslc:property>
      <oslc:Property>
        <oslc:name>parent</oslc:name>
        <oslc:propertyDefinition rdf:resource="http://md.kth.se/oslc/simulink/1.0/data#Block" />
      </oslc:Property>
    </oslc:property>
  </oslc:ResourceShape>
</rdf:RDF>
```
By analyzing the response, we can identify the attributes like uuid, name and type, and references like parent, children, ports and lines. For the references, referenced data type names are anticipated in a similar way. For example, reference ports is referenced to Port data type by extracting the last word of the URI http://localhost:8080/Simulink/shapes/Port of the property oslc:valueShape (line 53).
Also, we can obtain the numerical attributes of each attribute or reference by looking into the oslc:occurs attribute. According to the OSLC specification, the possibilities are Zero-or-one, Exactly-one, One-or-many and Zero-or-many.

**Step 5: Construct Data Resource Structures: Query Object Details**

If there’s no annotated oslc:ResourceShape given for a specific resource, we have to discover the structure of this resource on our own by querying a specific object in that data type. To begin with, we get the list of objects by following the URI of oslc:QueryCapability. Please note, here we assume we will get a list of resources with URIs instead of inlined resources.

In our example, if we pretend that we don’t have a oslc:ResourceShape for the Block resource, then we have to follow the URI http://localhost:8080/Simulink/block to get a list of Blocks:

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:oslc_simulink_data="http://md.kth.se/oslc/simulink/1.0/data#
  xmlns:owl="http://www.w3.org/2002/07/owl"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
  <rdf:Description rdf:about="http://localhost:8080/Simulink/block">
    <rdfs:member rdf:resource="http://localhost:8080/Simulink/block/1-fc429e1-a0e6-4316-b5bb-8a1f2630b87c"/>
    <rdfs:member rdf:resource="http://localhost:8080/Simulink/block/3-ce3ebf6-fb28-402d-99f8-c6556bb1504a"/>
    <rdfs:member rdf:resource="http://localhost:8080/Simulink/block/272fd601-9b74-4726-a4e3-22a57e451873"/>
  </rdf:Description>
</rdf:RDF>
```

Follow any one of the URI, we can get the details of one object. By analyzing the content of the response, we can also get the attributes or references of the current data type. The algorithm here is more complex, and is introduced later by an example. Also, we may need to query several different objects, to make sure we get a complete dump of all the properties.

In our example, if we look into the detail response of a Block object with URI http://localhost:8080/Simulink/block/272fd601-9b74-4726-a4e3-22a57e451873 (see line 10 above), we can see the object has attributes type, name, uuid and references ports with URI http://localhost:8080/Simulink/port/41a94277-3a02-4faf-a489-953c4b7fb28b (see line 9 below) and children with URI http://localhost:8080/Simulink/block/cae9c004-c635-429f-bde7-709b626c34f9 (see line 11 below). For the correct type of the references, we need to follow the referenced URI, and get the resource type by analyzing the response, just like we identify Block as the current object type by taking the local name of the whole resource (see
line 7 below). More queries of the current data type are needed, as we can see the result here is not complete compared to the result we obtained in Step 4.

```xml
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:oslc_simulink_data="http://md.kth.se/oslc/simulink/1.0/data#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
  <oslc_simulink_data:Block rdf:about="http://localhost:8080/Simulink/block/272fd601-9b74-4726-a4e3-22a57e451873"/>
  <oslc_simulink_data:lines rdf:resource="http://localhost:8080/Simulink/connection/5cedbac9-d627-4071-aa58-250e8a8dce41"/>
  <oslc_simulink_data:ports rdf:resource="http://localhost:8080/Simulink/port/41a94277-3a02-4faf-a489-953c4b7f28b"/>
  <oslc_simulink_data:children rdf:resource="http://localhost:8080/Simulink/block/cae9c004-c635-429f-bde7-709b626c34f9"/>
  <oslc_simulink_data:children rdf:resource="http://localhost:8080/Simulink/block/1cd858d5-fica-4943-852a-3190dd6f72a6"/>
  <oslc_simulink_data:type>SubSystem</oslc_simulink_data:type>
  <oslc_simulink_data:name>F1</oslc_simulink_data:name>
  <oslc_simulink_data:uuid>272fd601-9b74-4726-a4e3-22a57e451873</oslc_simulink_data:uuid>
</oslc_simulink_data:Block>
</rdf:RDF>
```

Please note, here we assume the name of the attributes or references is directly from the local name in the properties of the response, and the referenced data type is directly from the local name of the resource type following the resource URI.

**Step 6: Find Control Resources**

By analyzing the URIs of `oslc:creationFactory`, we can obtain a list of provided control resources. For the resource name, we assume it is from the URI of `oslc:creation`.

In our example, from the response of `oslc:ServiceProvider` with the URI `http://localhost:8080/Simulink/oslc/service_provider/control`, we can identify the control resources like `load_model`, `start_simulation`, `select`, etc (line 17, 24 31).

Please note, in the current demo, the discovery of the parameters for the control services is not available as no suitable directory service for parameters defined in OSLC.
6.2.3 Saving the Discovered Model

Saving the discovered model is easy with the help of EMF. Actually, the model stored in a file is equivalent to the *EPackage* object created in the memory, we can fill correct information to the object and serialize it to a file at last.

To facilitate the discovery algorithm, the *EPackage* object can be created before the discovery algorithm begins, and is used as a data structure to hold the information we discover in the process. For the initialization steps, two *EPackage* objects are created which represents the separate *data model* and the *core model* as we agreed, then sub-packages called *control* and *oslc* are created and added to the *core model*, and finally a special *Control class* is added to the *control sub-package*.

In the discovery process, updates including adding of new classes to the *data model*, adding of new *operations* to the *Control class* and adding of *attributes* or *references* to the data *classes* we created are made.

After the discovery, the two objects are written to the disk through EMF persistence API.

In our example, according to the model design and the result we discovered, the *data model* and *core model* are generated as in Figure 6.3 and Figure 6.4.

Compared to the model we designed and created manually in the construction process, as we agreed in the designing of the discovered tool adapter mode, here data classes are not inherited from the *Prototype* in the *ifest-common* model and more *oslc* annotations of the data or control services are annotated to facilitate the future use.

6.2.4 Model-based Generation of Code Stubs

The generation process here is quite similar to the generation process in the construction of tool adapters. The only difference is that here, the generation only involves with a few code stubs needed to integrate and consume the provided RESTful service.

According to SCA, to consume a RESTful web service, a Java interface should be available as the SCA service, and this is the only code we need to generate for the discovered tool adapter.

In our example, Java interface with the name `se.kth.md.generatedtoolchain.IDiscoveredSimulink` is created, for the content of this file, please refer to the generation result of the construction process in Appendix A as it is exactly the same as in the construction process.

In the big picture, an annotated object called `discoveredSimulink` typed in this interface is added to the main tool chain class. And in the configuration section of the tool chain, special configurations are added to bind the RESTful web service to the annotated object typed in the interface created.

Below is the configuration part of this discovered service in `<sca:component name="GeneratedToolChain">` section in the main composite file:
Figure 6.3. Discovered data model of Matlab/Simulink Tool Adapter
Figure 6.4. Discovered core model of Matlab/Simulink Tool Adapter
6.3 Summary of Assumptions

As in the construction process of tool adapters, the OSLC-based output only defines a specific way to output the information needed in OSLC. In the conversion from an EMF-based model to an OSLC-based output, information is lost inevitably. To fulfil the need to reconstruct the similar EMF based model for discovered services, assumptions are made to facilitate the discovery procedure. In Table 6.1, all the assumptions made in the discovery steps are summarized.
Table 6.1. Assumptions made in the automated discovery algorithm of unknown tool adapter conforming to OSLC

<table>
<thead>
<tr>
<th>Position</th>
<th>Assumptions made</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 2</td>
<td>Each data service includes an inlined <code>oslc:CreationFactory</code> resource which contains the URI for the POST method to create a new object and a <code>oslc:QueryCapability</code> resource which contains the URI for the GET of a list of objects (the two URIs are normally the same). Each control service only includes a inlined <code>oslc:CreationFactory</code> resource which contains the URI for the POST method to invoke this service.</td>
<td>Service type can be checked by checking if there’s any inlined <code>oslc:QueryCapability</code> resources.</td>
</tr>
<tr>
<td>Step 3</td>
<td>The name of the data resource is extracted from the last word of the URI of the inner property <code>oslc:resourceShape</code>, <code>oslc:resourceType</code>, <code>oslc:creation</code> or <code>oslc:queryBase</code> in the <code>oslc:CreationFactory</code> or <code>oslc:QueryCapability</code> property.</td>
<td>The simple algorithm to extract from the first available URI of the possible sources is used in the demo. More complex algorithm is discussed as well.</td>
</tr>
<tr>
<td>Step 3</td>
<td>The annotated <code>oslc:ResourceType</code> resource is given by providing an external URI</td>
<td>Alternative possibilities should be considered in the actual development.</td>
</tr>
<tr>
<td>Step 4</td>
<td>The name of the referenced data resource type is extracted from the last word of the URI of inner property <code>oslc:valueShape</code> or <code>oslc:valueType</code> in the corresponding section of the <code>oslc:ResourceShape</code></td>
<td>The simple algorithm is used in the model and more complex situations should be considered in the actual development.</td>
</tr>
<tr>
<td>Step 5</td>
<td>The return value of the query to GET a list of objects will be a list of resources with URIs instead of inlined resources.</td>
<td>Alternatives should be considered in the actual development.</td>
</tr>
<tr>
<td>Step 5</td>
<td>The name of the attributes or references are directly from the local name in the properties of the response of GET the details of one object, and the referenced data resource type is directly from the local name of the resource type following the referenced resource URI.</td>
<td></td>
</tr>
<tr>
<td>Step 6</td>
<td>Name of the control resource provided is from the URI of <code>oslc:creation</code> property inlined.</td>
<td>More complex situation should be considered in the actual development.</td>
</tr>
<tr>
<td>Step 3/6</td>
<td>the discovery of the (additional) parameters for the data or control services is not available as no suitable directory service for parameters defined in OSLC.</td>
<td></td>
</tr>
</tbody>
</table>
7. Conclusion

7.1 Summary

This thesis is a continuation of research on the tool adapters in tool integration with model-based approach. The original Matlab/Simulink tool adapter with only data functionalities is extended, in this basis, a more generic model-based tool adapter development platform is designed and implemented. With modeling, a lot of programming details are generalized and hidden, and can be fully understood by the users from different background who may not be proficient in programming. With modeling, tool adapters from different vendors can obey the same rule and be included in the tool chain with less configurations.

This thesis is part of the research in providing a model-based tool integration framework with the focus on tool adapters. The research mainly includes the construction and discovery of the tool adapters on the basis of the practices of extending the existing Matlab/Simulink tool adapter. After the implementation of the model-based tool adapter construction framework, the Matlab/Simulink tool adapter is rewritten to follow the same standard and as a validation of the construction framework and resource of the model-based tool adapter discover approach.

The whole work follows EMF as the base of tool adapter modeling and the model-to-tool-adapter generation process; follows OSLC to encapsulate data and control functionalities of the tools as different resources and be exposed according to the OSLC specification; follows SCA to encapsulate tool adapters in a separate composite, and be included as part of the SCA based tool chain. Guidelines of tool adapter modeling and discovered tool adapter modeling are proposed on the practices of OSLC and EMF, and with the assumptions on the modeling guidelines, model-to-tool-adapter generation engine and model-based discovery algorithm are created.

The result of the work can be used in the first step of creating tool adapters for different tools with the same standard, and in the last step of discovering tool adapters from other vendors and including them in the whole tool chain.

7.2 Limitations

As we summarized in Section 5.4 and 6.3, the major limitation of the work is as followed:

1. OSLC only specifies data resources clearly in it’s specification, thus to support the control functionalities, we make a strong assumptions on the separation
of the two categories of resources. Assumptions on the directory services for discovery are also made in the same point to facilitate the discovery process.

2. The parameters of data resources (as classes) cannot be presented well in EMF, thus cannot be included in the model-based construction framework as well as the discovery approach. This is also the reason that manual handing of parameters is required in the implementation of tool adapters.

3. Directory services for parameters are not specified in OSLC, thus can’t be included in the automatical discovery approach.

7.3 Future Work

The control functionalities are vital for the integration process, and should be included as part of OSLC. With the support of control integration functionalities, and clear specification on the handling of the different categories of services in the directory services, the construction framework as well as the discovery approach can be greatly improved. More sophisticated discovery algorithm can be implemented with more investigations of the resources we get through the interaction of the provided OSLC services.

With special agreement on parameter modeling for data resources, parameter handling can be fully generated in the construction framework, and a lot of human work can be saved. With the improvement of parameter handing in OSLC’s directory services, the discovery process can be more automated and much less work will be needed in including the discovered tool adapter as the whole tool chain.

Finally, the modeling of the tool adapter can be further studied to include more complex type of resources like embedded resources. With this basis, more work can be done in both the construction framework and discovery approach.
References


Appendix A.
Generated Code Stubs for Matlab/Simulink Adapter Based on the Tool Adapter Construction Platform

External Component/ISimulinkToolAdapter.java

```java
package se.kth.md.generatedtoolchain;

/**< Imports are omitted */

@Service
public interface ISimulinkToolAdapter extends IToolAdapter{

@Path("/block")
@GET
public Response GETBlockList(@Context MessageContext mc);

@Path("/block/{til_uuid}")
@GET
public Response GETBlock(@Context MessageContext mc);

@Path("/block")
@POST
public Response POSTBlock(@Context MessageContext mc);

@Path("/block/{til_uuid}")
@PUT
public Response PUTBlock(@Context MessageContext mc);

@Path("/block/{til_uuid}")
@DELETE
public Response DELETEBlock(@Context MessageContext mc);

 /**< Declaration of other data services are omitted here */

@Path("/control/load_model")
@POST
public Response LoadModel(@Context MessageContext mc);

 /**< Declaration of other control services are omitted here */

@Path("/oslc/service_provider_catalog")
@GET
@Produces({ "application/rdf+xml", "application/xml" })
```

68
public String GETServiceProviderCatalog(@Context MessageContext mc);

@Path("/oslc/service_provider/{service_provider_type}")
@GET
@Produces(["application/rdf+xml", "application/xml"])
public String GETServiceProvider(@Context MessageContext mc);

@Path("/shapes/Block")
@GET
@Produces(["application/rdf+xml", "application/xml"])
public String GETBlockShape(@Context MessageContext mc);

/* Declaration of other ResourceShape services are omitted here */
/* Administrative methods are omitted here */

package se.kth.md.generatedtoolchain;

/*@ Imports are omitted */
@Scope("COMPOSITE")
public class SimulinkToolAdapter extends ToolAdapter implements ISimulinkToolAdapter{

/*@ Administrative methods are omitted here */

@Reference
private ISimulinkHelper helper = null;

private ResponseBuilder getResponseBuilder() {
    return helper.getResponseBuilder();
}

/*@ Implementation details of the OSLC services are omitted here */

@Override
public Response GETBlockList(MessageContext mc) {
    return processTemplate(new RequestHolder(mc, false), "doGETBlockList");
}

@Override
public Response GETBlock(MessageContext mc) {
    return processTemplate(new RequestHolder(mc, false), "doGETBlock");
}

69
@Override
public Response POSTBlock(MessageContext mc) {
    return processTemplate(new RequestHolder(mc, true), "doPOSTBlock");
}

@Override
public Response PUTBlock(MessageContext mc) {
    return processTemplate(new RequestHolder(mc, true), "doPUTBlock");
}

@Override
public Response DELETEBlock(MessageContext mc) {
    return processTemplate(new RequestHolder(mc, false), "doDELETEBlock");
}

/*@ Implementation of other data services are omitted here */

@override
public Response LoadModel(MessageContext mc) {
    return processTemplate(new RequestHolder(mc, true), "doLoadModel");
}

/*@ Implementation of other control services are omitted here */

/*@ Implementation details of the OSLC services are omitted here */

@override
public Response GETServiceProviderCatalog(MessageContext mc) {
    Map<String, Object> bean = constructServiceProviderCatalog();
    return executeTemplate(new RequestHolder(mc, false), null, "spc.ftl",
                           bean);
}

@override
public Response GETServiceProvider(MessageContext mc) {
    RequestHolder rh = new RequestHolder(mc, false);
    String type = rh.getFirst("service_provider_type");
    ParamHelper.testRequiredParam("service_provider_type", type);
    Map<String, Object> bean = constructServiceProvider(type);
    return executeTemplate(rh, type, "sp.ftl", bean);
}

@override
public Response GETBlockShape(MessageContext mc) {
    return executeTemplate(new RequestHolder(mc, false), "data",
                           "shapes/Simulink/block.ftl", null);
}
package se.kth.md.generatedtoolchain.user;

@Service
public interface ISimulinkHelper {
    public static final String SERVICE_NAME = "Simulink";
    public ResponseBuilder getResponseBuilder();
    public String getNS(RequestHolder rh, String type);
    public IEvent doGETBlockList(RequestHolder rh);
    public IEvent doGETBlock(RequestHolder rh);
    public IEvent doPOSTBlock(RequestHolder rh);
    public IEvent doPUTBlock(RequestHolder rh);
    public IEvent doDELETEBlock(RequestHolder rh);
    public IEvent doLoadModel(RequestHolder rh);
}

package se.kth.md.generatedtoolchain.user;

@Service
public interface ISimulinkHelper {
    public static final String SERVICE_NAME = "Simulink";
    public ResponseBuilder getResponseBuilder();
    public String getNS(RequestHolder rh, String type);
    public IEvent doGETBlockList(RequestHolder rh);
    public IEvent doGETBlock(RequestHolder rh);
    public IEvent doPOSTBlock(RequestHolder rh);
    public IEvent doPUTBlock(RequestHolder rh);
    public IEvent doDELETEBlock(RequestHolder rh);
    public IEvent doLoadModel(RequestHolder rh);
}

package se.kth.md.generatedtoolchain.user;

@Service
public interface ISimulinkHelper {
    public static final String SERVICE_NAME = "Simulink";
    public ResponseBuilder getResponseBuilder();
    public String getNS(RequestHolder rh, String type);
    public IEvent doGETBlockList(RequestHolder rh);
    public IEvent doGETBlock(RequestHolder rh);
    public IEvent doPOSTBlock(RequestHolder rh);
    public IEvent doPUTBlock(RequestHolder rh);
    public IEvent doDELETEBlock(RequestHolder rh);
    public IEvent doLoadModel(RequestHolder rh);
}

package se.kth.md.generatedtoolchain.user;

@Service
public interface ISimulinkHelper {
    public static final String SERVICE_NAME = "Simulink";
    public ResponseBuilder getResponseBuilder();
    public String getNS(RequestHolder rh, String type);
    public IEvent doGETBlockList(RequestHolder rh);
    public IEvent doGETBlock(RequestHolder rh);
    public IEvent doPOSTBlock(RequestHolder rh);
    public IEvent doPUTBlock(RequestHolder rh);
    public IEvent doDELETEBlock(RequestHolder rh);
    public IEvent doLoadModel(RequestHolder rh);
}
public class SimulinkHelper implements ISimulinkHelper {

    @Reference
    private IIdManager idManager = null;

    @Reference
    private IConverterService converterService = null;

    @Init
    public void initComponent() {
        // Start of user code SimulinkHelper initComponent
        // TODO Register idManager with IInitIdService
        // idManager.init(SERVICE_NAME, this);
        // TODO Register XMI parser if needed
        // Resource.Factory.Registry.INSTANCE.getExtensionToFactoryMap().put(
        //     Resource.Factory.Registry.DEFAULT_EXTENSION,
        //     new XMIResourceFactoryImpl());
        // TODO Register user meta model
        // ResourceSet resourceSet = new ResourceSetImpl();
        // resourceSet.getPackageRegistry().put(Ifest_commonPackage.eNS_URI,
        //     Ifest_commonPackage.eINSTANCE);
        // resourceSet.getPackageRegistry().put(OSLCSimulinkDataPackage.eNS_URI,
        //     OSLCSimulinkDataPackage.eINSTANCE);
        // End of user code
    }

    @Override
    public String getNS(RequestHolder rh, String type) {
        // Start of user code SimulinkHelper getNS
        // TODO Implement getNS
        // if (type.equalsIgnoreCase("control"))
        //     return ControlPackage.eNS_URI;
        // else
        //     return OSLCSimulinkDataPackage.eNS_URI;
        // return 
        // End of user code
    }

    private IEventCustomResponse processRequest(String handlerClassName, IHandlerData handlerData) {
        // Start of user code SimulinkHelper user methods
        // End of user code
    }
}
IQuery query = QueryFactory.INSTANCE.createQuery(handlerClassName, handlerData);
IEvent event = EventFactory.INSTANCE.createOslcEvent(this, query);

// Start of user code SimulinkHelper processRequest
// TODO process(event) and set the corresponding part of the event
return new OslcEventCustomResponse(event);
// End of user code

private IHandlerData prepare(RequestHolder rh) {
    IHandlerData handlerData = new HandlerData();
    // Start of user code SimulinkHelper prepare RequestHolder
    // TODO identify the service param key for the current tool, a default service param could be set
    rh.setServiceParam("model_name", "york");
    ParamHelper.addParam(handlerData, "model_name", rh.getServiceParam());
    // Comment the following line to disable the multi subtree access
    ParamHelper.addParam(handlerData, "subtree_levels", Integer.valueOf(rh.getSubtreeLevels().toString()));
    // End of user code
    return handlerData;
}

/*@Override*/
public IEventCustomResponse doGETBlockList(RequestHolder rh) {
    IHandlerData handlerData = prepare(rh);
    // Start of user code SimulinkHelper GETBlockList Impl
    // Add custom params, validate the params
    // Do in this way to add the param
    // ParamHelper.addParam(handlerData, "PARAM_KEY", rh);
    // Do in this way to add the resolved id
    // ParamHelper.addParam(handlerData, "PARAM_KEY", resolve(rh, "PARAM_KEY"));
    // Do in this way to validate the param
    // ParamHelper.testRequiredParam("PARAM_KEY", rh);
    return processRequest("model.get.Blocks", handlerData);
    // End of user code
SCA configuration file/Simulink.composite

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<sca:composite xmlns:sca="http://www.osoa.org/xmlns/sca/1.0"
  xmlns:frascati="http://frascati.ow2.org/xmlns/sca/1.1"
  xmlns:id_manager="se/kth/md/id_manager"
  xmlns:converter="se/kth/md/converter"
  xmlns:frascati="http://frascati.ow2.org/xmlns/sca/1.1"
  name="se.kth.md.generatedtoolchain.Simulink"/>
  
  <sca:component name="IdManager">
    <sca:implementation composite name="id_manager:IdManager.composite"/>
    <sca:service name="IdManager">
      <sca:interface java interface="se.kth.md.id_manager.IIdManager"/>
    </sca:service>
  </sca:component>

  <sca:component name="converter">
    <sca:implementation composite name="converter:Converter.composite"/>
    <sca:service name="RDFConverterService">
      <sca:interface java interface="se.kth.md.converter.IConverterService"/>
    </sca:service>
    <sca:service name="XMLConverterService">
      <sca:interface java interface="se.kth.md.converter.IConverterService"/>
    </sca:service>
    <sca:service name="AutoConverterService">
      <sca:interface java interface="se.kth.md.converter.IConverterService"/>
    </sca:service>
  </sca:component>

  <sca:component name="Simulink">
    <sca:implementation java class="se.kth.md.generatedtoolchain.SimulinkToolAdapter"/>
    <sca:service name="ISimulinkToolAdapter">
      <frascati:binding rest uri="http://localhost:8080/"/>
    </sca:service>
  </sca:component>

  <sca:component name="SimulinkHelper">
    <sca:implementation java class="se.kth.md.generatedtoolchain.user.SimulinkHelper"/>
  </sca:component>
```

<sca:component>
  <sca:service name="SimulinkService" promote="Simulink/" promote="SimulinkService">
    <sca:interface java interface="se.kth.md.generatedtoolchain.ISimulinkToolAdapter" />
  </sca:service>
  <sca:wire source="SimulinkHelper/idManager" target="IdManager/" target="IdManager" />
  <sca:wire source="SimulinkHelper/converterService" target="converter/AutoConverterService" />
  <sca:wire source="Simulink/helper" target="SimulinkHelper/" target="ISimulinkHelper" />
</sca:component>